Lecture Notes in Computer Science 2590
Edited by G. Goos, J. Hartmanis, and J. van Leeuwen

Springer
Berlin
Heidelberg
New York
Barcelona
Hong Kong
London
Milan
Paris
Tokyo

Stéphane Bressan Akmal B. Chaudhri
Mong Li Lee Jeffrey Xu Yu
Zoé Lacroix (Eds.)

Efficiency and Effectiveness of XML Tools and Techniques and Data Integration over the Web

VLDB 2002 Workshop EEXTT
and CAiSE 2002 Workshop DIWeb
Revised Papers

Springer

Volume Editors

Stéphane Bressan
Mong Li Lee
School of Computing, NUS, Department of Computer Science
3 Science Drive 2, Singapore 117543, Republic of Singapore
E-mail:{steph/leeml}@comp.nus.edu.sg

Akmal B. Chaudhri
IBM developerWorks
6 New Square Bedfont Lakes, Feltham, Middlesex TW14 8HA, UK
E-mail: akmal.b.chaudhri@uk.ibm.com

Jeffrey Xu Yu
Chinese University of Hong Kong
Dept. of Systems Engineering and Engineering Management
Shatin, New Territories, Hong Kong
E-mail: yu@se.cuhk.edu.hk

Zoé Lacroix
Arizona State University, Mechanical Aerospace Engineering
P.O. Box 876106, Tempe, AZ 85287-6106, USA
E-mail: zoe.lacroix@asu.edu

Cataloging-in-Publication Data applied for

A catalog record for this book is available from the Library of Congress

Bibliographic information published by Die Deutsche Bibliothek
Die Deutsche Bibliothek lists this publication in the Deutsche Nationalbibliografie;
detailed bibliographic data is available in the Internet at <http://dnb.ddb.de>.

CR Subject Classification (1998): H.4, H.2, H.3

ISSN 0302-9743
ISBN 3-540-00736-9 Springer-Verlag Berlin Heidelberg New York

Springer-Verlag Berlin Heidelberg New York
a member of BertelsmannSpringer Science+Business Media GmbH

http://www.springer.de

Typesetting: Camera-ready by author, data conversion by DA-TeX Gerd Blumenstein
Printed on acid-free paper SPIN 10872475 06/3142 5 4 3 2 1 0

Preface

This volume comprises papers from the 1st VLDB Workshop on Efficiency and Effectiveness of XML Tools and Techniques (EEXTT 2002) and selected papers from the 2nd Workshop on Data Integration over the Web (DIWeb 2002).

Efficiency and Effectiveness of XML Tools and Techniques (EEXTT)

As XML emerges as the standard for data representation and interexchange on the World-Wide Web, a variety of XML Management Systems (XMLMS) differing widely in terms of expressive power and performance are becoming available. The majority of these systems are legacy systems, that is, relational database systems which have been extended to load, query, and publish data in XML format. A few are native XMLMS and capture almost all the characteristics of XML data representation. A large number of new techniques are being devised for the management of XML data. The EEXTT workshop focused on the evaluation of the efficiency, effectiveness, and performance of XML management systems, tools, and techniques.

This first workshop was organized in conjunction with the 28th International Conference on Very Large Data Bases, held in Hong Kong, China. The workshop Call for Papers (CFP) solicited contributions covering:

- Storage of XML data
- Compression of XML data
- Security for XML data
- Generation of XML data from legacy applications
- Indexing and retrieval of XML data
- XML query languages
- Query processing over XML data
- Interchange and integration of XML data
- Benchmarks for the management of XML data

After peer review, nine papers were selected for presentation. These were grouped into three main areas: languages, modeling and integration, and storage. A special invited session on benchmarking XML was also organized. This included papers presented by representatives from major research groups around the world working on performance benchmarks for XML tools and applications.

For this LNCS volume, all papers were revised based upon reviewer feedback and questions and issues raised during the workshop.

Workshop Program Committee:

Zohra Bellahsène	LIRMM (France)
Elisa Bertino	University of Milan (Italy)
Timo Böhme	University of Leipzig (Germany)
Philippe Bonnet	University of Copenhagen (Denmark)
Stéphane Bressan	National University of Singapore (Singapore)
Akmal B. Chaudhri	IBM developerWorks (USA)
Gillian Dobbie	University of Auckland (New Zealand)
Wynne Hsu	National University of Singapore (Singapore)
Tok Wee Hyong	Singapore Telecommunications (Singapore)
Zoé Lacroix	Arizona State University (USA)
Mong Li Lee	National University of Singapore (Singapore)
Ioana Manolescu	INRIA (France)
Ullas Nambiar	, Arizona State University (USA)
Michael Rys	Microsoft Corporation (USA)
Zahir Tari	RMIT University (Australia)
Jeffrey Xu Yu	Chinese University of Hong Kong (Hong Kong)

Data Integration over the Web (DIWeb)

Many approaches have been developed in the past to address data integration over distributed heterogeneous databases. However, the exploitation of Web data raises multiple new challenging issues. Mediation or multidatabase systems integrate many data sources generally using the notion of views to make them accessible through an integrated schema by a unique system. Most of these approaches are designed to integrate databases not Web data. But is the Web a database? Are the data sources available on the Web databases? Unlike data stored in databases, Web data is semistructured and its structure is both explicit and implicit. Access to Web data is performed through rather limited access, such as cgi forms, browsing, and extraction, as well as other applications such as search engines. The limitation and the variety of accesses to Web data further distinguish the Web from a database. In addition, access to Web data is costly, uncertain and unreliable when compared to database systems.

The 1st International Workshop on Data Integration over the Web, organized by Zohra Bellahsène, was held in conjunction with the International Conference on Advanced Information Systems Engineering (CAiSE) in Interlaken, Switzerland, in June 2001. In May 2002, DIWeb was again held in conjunction with CAiSE and took place in Toronto, Canada. The second DIWeb focused on XML, complex applications, and the Semantic Web. An invited talk by Alberto Mendelzon (University of Toronto), presented Tox, an XML management system. Several contributions covered various uses of XML, such as an XML data exchange language, a case tool to design XML views, and a caching technique to optimize XPath execution. Three papers addressed issues related to the integration of the Web for specific applications: scientific data, printed and digital data, and geographical data. Issues related to the Semantic Web were addressed in four

papers. Presented approaches included exploiting Web services, source capabilities, and a variety of explicit and implicit knowledge for Web sites and schema mapping. The workshop concluded with a discussion animated by Tamer Özsu (University of Waterloo), Joe Wigglesworth (Centre for Advanced Studies, IBM Toronto Lab), Louiqa Raschid (University of Maryland), and Anthony Tomasic (XML Media) during a panel entitled "Using Standards for Data Integration over the Web."

DIWeb 2002 lasted an entire day including an invited talk, ten presentations of scientific papers (accepted after peer-review), an invited paper, and a final panel. Five papers were selected to be included in this issue of Lecture Notes in Computer Science.

Workshop Program Committee:

Bernd Amann	INRIA (France)
Omar Boucelma	Université de Provence (France)
Stéphane Bressan	National University of Singapore (Singapore)
Subbarao Kambhampati	Arizona State University (USA)
Brigitte Kerhervé	Université du Québec (Canada)
Mong Li Lee	National University of Singapore (Singapore)
Bertram Ludäscher	San Diego Supercomputer Center (USA)
Alberto Mendelzon	University of Toronto (Canada)
Vincent Oria	NJ Institute of Technology (USA)
Louiqa Raschid	University of Maryland (USA)
Mark Roantree	Dublin City University (Ireland)
Shalom Tsur	BEA Systems (USA)

Thanks also to the following additional reviewers: Ilkay Altintas, Chaitan Baru, Ajay Hemnani, Amarnath Gupta, Zaiqing Nie, Ullas Nambiar, Eli Rohn, and Sreelakshmi Vaddi.

The organizers of both workshops would like to thank all program committee members, as well as all external referees, for their excellent work in evaluating the submitted papers. We are also very grateful to the organizers of VLDB and CAiSE for their help and support. Finally, our sincere thanks to Mr. Hofmann at Springer-Verlag for his guidance and enthusiasm in putting this volume together.

London, December 2002

Stéphane Bressan, Akmal B. Chaudhri,
Mong Li Lee, Jeffrey Xu Yu
(Program Chair and Co-chairs EEXTT)
Zoé Lacroix
(Program Co-chair DIWeb)

Table of Contents

Efficiency and Effectiveness of XML Tools and Techniques (EEXTT)

XML Languages

XML Modeling and Integration

XML Storage

Benchmarking XML

Data Integration over the Web (DIWeb)

A Proposal for an XML Data Definition and Manipulation Language

Dare Obasanjo[1] and Shamkant B. Navathe[2]

[1] 25hoursaday.com
kpako@yahoo.com
[2] College of Computing
Georgia Institute of Technology, Atlanta, GA 30332-0280
sham@cc.gatech.edu

Abstract. XML has become a popular data interchange and storage format, which in recent times has precipitated the rise of XML-enabled relational databases as well as native XML databases. This paper outlines a data definition and manipulation language for XML repositories that enables users to perform data management tasks such as creation and deletion of indices, collections and documents. The language proposed also provides the ability to perform queries, transformations and updates on the documents in the XML repository either singly or across an entire collection. A syntax for the language is presented as extensions to the W3C's XML Query language (XQuery) and also as a new language with syntax borrowed heavily from SQL for the relational model and DL/1 of IBM's IMS system for the hierarchical model. A prototype implementation of the language has been partially completed.

1 Introduction

Currently it is not uncommon to see a native XML or XML-enabled database that uses XPath for querying, an XML-based language or XML Document Object Model (DOM) interaction for updates, and a graphical or command line interface for managing documents and collections. This highlights the need for providing a single mechanism that allows programmers and ordinary users to manipulate documents and collections within XML database systems.

The Simple XML Data Manipulation Language (SiXDML) was designed to create common syntax and semantics for performing tasks most often required of XML repositories. SiXDML has two potential syntaxes: (i) extensions to the W3C XQuery language or (ii) as a brand new language with its keywords borrowed from older data manipulation languages.

SiXDML is designed to be easily understood by programmers and non-programmers alike, internet-aware, and combine a minimum of complexity with a maximum of functionality while being straightforward to implement. SiXDML aims to plug the hole that currently exists for a data manipulation language that allows one to

S. Bressan et al. (Eds.): EEXTT and DIWeb 2002, LNCS 2590, pp. 1-21, 2003.

query and update XML documents while also providing a means to perform database management activities. Although numerous query languages exist for XML including Lorel, Quilt, UnQL, XDuce, XML-QL, XPath, XQL, XQuery and YaTL yet no popularly accepted XML query language possesses data modification semantics such as delete, replace and insert that exist in a relational data manipulation language like SQL [11] or a hierarchical data manipulation language like DL/I [9], [10].

XQuery which seems set to become the standard query language for XML repositories does not have update semantics in the December 2001 draft [2] although some investigation on how to add updates to XQuery have been undertaken by researchers like Lehti [4] and Rys, Florescu, Robie and Chamberlin [1]. The XQuery working group has hinted that a version of XQuery that contains updates is about a year or more away. From statements made by a member of the XQuery Working Group (WG), the primary focus of the XQuery WG will be creating a robust type system that allows static type checking upon which updates can then be built. Static type checking would enable one to determine whether the XML generated by a query conformed to a particular schema. Type checking differs from validation in that validation occurs after the query has generated its results while type checking involves analysis of the query before it is executed. The XQuery static type system is based on type inference. Type inference is fallible because it cannot typically infer the most specific type (or schema) for the results of a query or query expression. Thus some researchers question the viability of a static type system based on type inference because of the possibility of valid queries being rejected [5], [6].

However, proposals for a language that also allows one to manage indices, collections and schemata in an XML repository in the same manner one can manage indexes, triggers and relations in a relational database with SQL have not been forthcoming.

SiXDML is designed with familiarity in mind and thus borrows heavily from the syntax of SQL and DL/I for its keywords; these languages have been available since the mid 1970s and mid 1960s respectively and therefore have been widely accepted in the database community. Similarly, SiXDML uses XPath for performing queries due to its widespread popularity and relative simplicity [2]. Since one of the goals of SiXDML is to provide a language that is easily understood by programmers and non-programmers alike, programming constructs like variable declarations, bound variables, functions, and nested queries have been avoided. The combination of using XPath for queries and the ability to pipe the results of queries to a filter such as an XSLT stylesheet or an XQuery expression makes the effects of a lack of sophisticated programming constructs in SiXDML minimal. An XML syntax for SiXDML, similar to that used for XUpdate [8] and XQueryX [19], was considered and then discarded for a number of reasons. The primary reason being that an XML syntax would have added to the complexity of the language by making it unnecessarily verbose as well as awkward for users of the language with a background in relational, hierarchical or object oriented database systems to adopt.

The syntax for the proposed XQuery extensions that will constitute SiXDML is based on that in [1]. However, at the current time no proposals exist for a language that integrates a data definition language (DDL) with a query language and a data manipulation language (DML).

The remainder of this paper is organized as follows. The next section provides an overview of some of the concepts related to XML repositories and highlights the problems that SiXDML was designed to solve. Section 3 gives an overview of the problems facing XML repositories and how SiXDML solves some of these problems. A section that explains the syntax and semantics of SiXDML statements follows this. Section 4 covers the grammar for both proposed SiXDML syntaxes while the final section outlines an API similar to the ODBC and JDBC APIs that enable clients to execute SiXDML statements.

2 Concepts and Terminology

In this section, some concepts and terminology from the realm of XML databases, which are a basic part of the framework upon which SiXDML is built, are described.

2.1 Document

A document is considered to be a well-formed and optionally valid XML document that can be stored in an XML repository.

2.2 Collection

A collection contains a group of XML documents, which may optionally conform to a single schema and can be queried and updated simultaneously by a single SiXDML statement. To SiXDML, a collection is similar to a directory on a file system that contains XML documents. A homogenous collection whose documents conform to one or more schemata can be considered to be similar to a row in relational table.

2.3 Schema

A schema is a document that is used to specify constraints on the XML documents in a particular collection. SiXDML is not designed with any specific schema technology given special consideration and instead will rely upon implementations to specify what schema technologies will be used or supported. Thus DTDs [13], W3C XML schema [14], XDR [20] files and RELAX NG [21] documents can all be supported by an XML repository that uses SiXDML.

2.4 Index

In many XML repositories, query performance can be improved by specifying certain elements, attributes or access patterns that should be indexed. Different repositories have different indexing capabilities; some like eXcelon's XIS allow one to index each individual word in an element (via a text index), the text value of each element (via a value index), or the expected node structure that will be returned by the query (via a structure index).

The design of SiXDML takes into consideration the fact that many different indexing techniques for XML data exist. Section 4.2 has further details.

2.5 Built-in Functions

To utilize some aspects of SiXDML as extensions to the XQuery language, certain functions will need to be added to the XQuery function library. For instance, a function similar to the XQuery document function called collection will be needed to retrieve a collection containing XML documents or subcollections.

All functions that are assumed to be a part of XQuery for the purpose of SiXDML will be treated as built-ins. Their function names will be prefixed "sixdml" which is considered to be bound to the SiXDML namespace:

"xmlns://www.sixdml.org/2002/04/language/".

2.6 XML Repository

An XML repository is a storage system, preferably one that supports ACID transactions [11], that manages XML documents and controls access to them. Native XML databases, XML content management systems and XML-enabled databases are all examples of XML repositories.

2.7 Query Processor

The query processor accepts SiXDML syntax, parses it and executes the query against the XML repository. It is expected that the query processor is capable of performing optimizations to improve the performance of queries before executing them.

3 Language Requirements

The W3C has specified requirements that XQuery language must meet to satisfy its design goals [12]. Similarly Lehti's XQuery update proposal [4] describes certain requirements for a language for updating XML. This section combines a number of these ideas to create requirements for an XML data definition language and data manipulation language for XML repositories. The points below use the words *must*, *should*, and *may* as used in several XQuery requirements document:

- An XML DDL/DML *may* have more than one syntax binding. At least one query language syntax *must* be easy for humans to read and write.
- An XML DDL/DML *should* consist of language elements users are familiar with from past experience.
- An XML DDL/DML *should* support retrieval of remote documents in the process of executing operations.
- An XML DML *must* not enforce any particular query evaluation strategy.
- An XML DDL/DML *must* define standard error conditions that can occur during query execution.

- An XML DML *must* support the ability to query and update single documents.
- An XML DML *should* support the ability to query and update collections of documents in a single operation.
- An XML DML *should* be able to combine related information from different parts of a given document or from multiple documents.
- An XML DDL/DML *must* be namespace aware.
- An XML DML *must* be able to recognize and select all of the XML node types specified in the XML 1.0 recommendation [13].
- An XML DDL/DML *must* not be tied to a specific XML schema technology but instead *must* support schema validation generically.
- An XML DML queries *must* be possible whether or not a schema is available.
- AN XML DML *must* provide access to the schema for a document, if one exists.
- An XML DDL *should* support the ability to create, manipulate and delete indexes on documents and collections of documents.
- An XML DDL *must* provide a way to constrain the members of a collection of documents by using a schema.
- An XML DDL/DML *must* support the ability to create, manipulate and delete collections of documents.
- An XML DML *must* be able to create new XML nodes within a query for use in updates.
- An XML DML *must* be able to insert XML nodes into any position in a document.
- An XML DML *must* be able to change values of elements and attributes without creating new nodes.
- An XML DML *must* be able to delete any XML node or subtree from a document.
- An XML DML *should* be able to rename elements and attributes.
- An XML DML *should* be able to alter namespace information in a document.
- An XML DDL/DML *may* allow for the creation of persistent named queries analogous to SQL views and stored procedures.
- An XML DML *may* utilize shortcuts to more often used sequences of insert, rename and delete operations such as replace and move.
- An XML DDL/DML *must* not contain any constructs whose semantics preclude usage in database systems that support transactions with ACID properties.

4 Syntax and Semantics of SiXDML

The following section describing the semantics for SiXDML presents dual syntaxes for each statement. One set of statements is for SiXDML as an extension to XQuery and the other for a brand new language that utilizes an SQL-like syntax. In the alternate syntax, language tokens are placed in quotes.

4.1 Collection Management

A collection is similar to a directory on a file system or to a relational table in that its content can be queried or updated in a single operation. Since like documents are often stored together in a collection, SiXDML supports constraining the contents of collections using schemas. Thus each document in the collection has to be valid against a specified schema. Schema constraints on collections also provide information that can be used for optimization (see section 4.3).

In SiXDML, each collection shall have an attached XML document containing metadata about the collection, which can be manipulated via SiXDML operations. The root node of a collection's metadata document is a <sixdml:metadata> element which cannot be deleted although no such restrictions apply to its children. The metadata document contains information about the schema and indexes that pertain to the collection. The W3C XML schema for the metadata document for a collection in SiXDML is shown below.

```
<xs:schema targetName-
space="xmlns://www.sixdml.org/2002/04/language/"
xmlns:sixdml="xmlns://www.sixdml.org/2002/04/language/"
elementFormDefault="qualified"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
 <xs:element name="metadata">
  <xs:complexType>
   <xs:all>
    <xs:element ref="sixdml:schema" />
    <xs:element ref="sixdml:indexes" />
   </xs:all>
  </xs:complexType>
 </xs:element>
 <xs:element name="indexes">
  <xs:complexType>
   <xs:sequence>
<xs:element name="index" minOccurs="0"
   maxOccurs="unbounded">
     <xs:complexType>
      <xs:sequence>
       <xs:element name="name" type="xs:string" />
       <xs:element name="type" type="xs:string" />
       <xs:element name="key" type="xs:string" />
       <xs:any namespace="##any" processContents="lax" minOc-
curs="0"
         maxOccurs="unbounded" />
      </xs:sequence>
     </xs:complexType>
    </xs:element>
   </xs:sequence>
   <xs:attribute name="collection" type="xs:string" />
  </xs:complexType>
 </xs:element>
```

```
 <xs:element name="schema">
  <xs:complexType mixed="true">
   <xs:sequence>
    <xs:any namespace="##any" minOccurs="0" maxOc-
curs="unbounded" processContents="skip"/>
   </xs:sequence>
  </xs:complexType>
 </xs:element>
</xs:schema>
```

Manipulating the metadata for a collection by performing DML operations on a virtual XML document although simpler for the end users may be difficult for some XML repositories to implement. In such cases, DDL operations with the semantics of those proposed in the alternate syntaxes in sections 4.1.2, 4.2.2, 4.3.2, 4.4.2, and 4.5.2 should be considered as alternatives.

Although some native XML databases can support nested collections, it is unlikely that XML-enabled relational databases will be able to support this feature without some difficulty. SiXDML is designed to work with both hierarchical and flat collection models.

4.1.1 XQuery Syntax for Collections Management

A collection in an XML repository can be considered to be a persistent, named, sequence of document nodes in XQuery parlance. To access collections, the built-in XQuery function called xf:collection which returns a sequence of document nodes is utilized. For the creation and deletion of collections, built-in functions are used as well. The signature of the functions for creating and deleting collections respectively are as follows:

```
sixdml:create_collection(string $uri) => boolean
sixdml:drop_collection(string $uri) => boolean
```

where the *$uri* parameter for both the sixdml:create_collection and sixdml:delete_collection functions has the same semantics as in the xf:collection function [22]. The sixdml:create_collection function returns true if the collection was successfully created and false if it already exists. The sixdml:drop_collection function returns true if the collection exists and was deleted but false if the collection does not exist.

It is also desirable to create a collection that is constrained by a specific schema thus making all its members documents of the same type. The signature for the create_constrained_collection function is as follows:

```
sixdml:create_constrained_collection(string schema_uri,
string uri) => boolean
```

where the schema_uri parameter is the URI where the schema can be found.

It should be noted that since XQuery is designed to be a functional language and the aforementioned functions have side effects they cannot be freely intermixed with other XQuery functions without awareness of the ramifications.

Any errors that occur in the processing of any of the aforementioned functions such as the $uri parameter not being a valid member of the anyURI lexical space [23] or an inability to create or delete a collection returns the error value as defined in [17].

4.1.2 Alternate Syntax for Collection Management

```
CreateStatement ::= "create" CreateCollectionClause
CreateCollectionClause ::= CollectionClause("constrained by"
Url)?
CollectionClause ::= 'collection' CollectionPath
CollectionPath ::= Nmtoken('/' Nmtoken)*
```

The create collection statement creates a collection with the specified path and optionally specifies the schema that will be used to constrain the documents in the collection. The schema can be located over the internet, in a local schema cache or on a local filesystem. If the schema cannot be located then the collection is not created. Below are some examples of the create collection statement.

```
create collection bookstore
create collection bookstore/out_of_stock
```

The first statement creates a collection at the top level of the repository named "bookstore" while the second creates a subcollection of the bookstore collection named "out_of_stock". Either statement fails if the collection being created already exists. Create collection statements that attempt to create a subcollection in a non-existent parent collection also end in failure.

```
create collection bookstore constrained by
http://www.25hoursaday.com/books.xsd
```

The above statement creates a collection at the top level of the repository named "bookstore" and specify that only documents that validate against the schema obtained from http://www.25hoursaday.com/books.xsd can be stored in the collection. This statement fails if the bookstore collection already exists or if the schema could not be retrieved.

```
DropStatement ::= "drop" CollectionClause
```

The drop collection statement deletes a collection and all its children from the XML repository. Below is an example of a drop collection statement.

```
drop collection bookstore
```

The above statement deletes the bookstore collection and any documents or subcollections that were contained within it.

```
ShowStatement ::= 'show' CollectionClause
```

The show collection statement lists the contents of the collection, which should typically be documents and/or subcollections. Below is an example of a show collection statement.

```
show collection bookstore
```

The above statement lists the names of the documents and subcollections in the bookstore collection.

4.2 Index Management

A wide variety of indexing options are supported by the current generation of XML database products. Indexing text in element nodes, attribute values, document structure and more are all valid options for an XML repository. Thus an XML data definition language that supports indexes must be flexible in how it enables the XML repository to specify indexes. In SiXDML only a collection of documents can have indexes specified over it.

The schema shown in section 4.1 requires certain information that should be common across all indexes across implementations of XML repositories namely the name, type and key fields for the index. Room is also allowed for vendor specific index information to be added to the XML representation of the index.

4.2.1 XQuery Syntax for Index Management

The index information about a collection is stored in its associated metadata document described in section 4.1.1. Below is an example of querying the names of indexes that apply to a collection and their key fields from the metadata for the collection's metadata.

```
<idx-key-val-pairs>{
  for $idx in
    collection("mydb:/root-
collection")/metadata/indexes/index,
  $idxname in $idx/name,
  $idxkey in $idx/key
return
  <idx-key-val-pair>
   <name>{$idxname/text()}</name>
    <key>{$idxkey/text()}</key>
  </idx-key-val-pair>
}
<idx-key-val-pairs>
```

The index information can be updated using the DML operations (see section 4.5).

4.2.2 Alternate Syntax for Index Management

```
CreateStatement ::= 'create' CreateIndexClause
CreateIndexClause ::= IndexClause 'of type' Nmtoken 'with key
=' XPathExpr(',' KeyValuePair)* 'on ' CollectionClause
IndexClause ::= 'index' Nmtoken
KeyValuePair ::= Nmtoken '=' (Char)+
```

The create index statement creates an index on a particular attribute or element for all documents in a collection. The key is the value that will be searched against while the optional list of key value pairs contains parameters that are dependent on the type of index which will vary between XML repositories. The XPath expression used as the value of the index key is a subset of XPath 1.0 equivalent to the one used by XML Schema for identity constraints [14]. Specifically, the XPath for the index key must evaluate to one or more element or attribute nodes.

Due to the varying indexing strategies that can be used by XML repositories, the SiXDML language does not enumerate the kind of indexes that can be used with the create index statement. Below is an example of a create index statement.

```
create index title-index of type value_index with key=/book,
ELEMENT=/book/@title on collection bookstore
```

The above statement creates an index named “title-index” that optimizes searches that utilize the value of the title attribute of a <book> element. It is assumed that the XML repository has an index type named “value_index”.

```
DeleteStatement ::= ‘delete’ IndexClause ‘from’ Collection-
Clause
```

The delete index statement removes an index that applies to a particular collection. An example of a delete index statement is shown below.

```
delete index val-index from collection bookstore
```

The above statement removes the value index named val-index that applies to every document in the bookstore collection.

```
ShowStatement ::= ‘show (indexes | index Nmtoken) on’ Collec-
tionClause
```

The show indexes statement displays the indexes that apply to a collection as XML that conforms to the schema described in section 4.2. XML repositories can transform the XML to make it more palatable to human readers. The show index statement has similar semantics. An example of a show indexes statement is shown below.

```
show indexes on collection bookstore
```

4.3 Schema Management

The ability to constrain the contents of an XML repository via a schema technology has several benefits. The primary benefit from a user’s standpoint is that the structure of the documents stored in the XML repository can be guaranteed to conform to a particular format. The primary benefit to the XML repository is that optimizations can be performed if certain knowledge is known up front such as whether a field will always be an integer within a certain range or the maximum length of particular string value.

There exist a number of XML schema technologies such as DTD, XDR, W3C XML Schema, Relax NG, and a number of others. SiXDML was designed to support multiple schema technologies thus preventing the need for users to alter their existing schema technology in order to use SiXDML. However an implementation of SiXDML as extensions to the XQuery language will need to support W3C XML schema.

A schema for a collection can be specified during or after creation of the collection. If an attempt is made to constrain a collection after documents have been inserted into the collection then each document in the collection must be valid against the schema or the operation will fail. Similarly updates to a collection’s schema that invalidate documents currently in the collection will also fail.

4.3.1 XQuery Syntax for Schema Management

The schema for a collection is stored in its associated metadata document described in section 4.1. The schema(ta) for the collection can be found under the

<sixdml:schema> element of the root <sixdml:metadata> element. The purpose of the <sixdml:schema> element is as a wrapper which disguises the fact that a collection may have a non-XML schema or multiple schemata. Below is an example of querying a collection for its associated W3C XML schema .

```
namespace xs = http://www.w3.org/2001/XMLSchema
namespace s = "xmlns://www.sixdml.org/2002/04/language/"
collection("mydb:/root-
collection")/s:metadata/s:schema/xs:schema
```

DTDs and other non-XML schemata can be stored in CDATA sections under the <sixdml:schema> child of the <sixdml:metadata> element.

The schema for a collection can be updated using the DML operations (see section 4.5).

4.3.2 Alternate Syntax for Schema Management

```
ConstrainStatement ::= 'constrain' CollectionClause 'with'
URL
```

The constrain collection statement specifies the schema that will be used to constrain a collection. Every document in the collection must validate against the schema or the operation fails. If there is already a schema constraining this collection it must be replaced if all documents validate against the new schema. An example of a constrain collection statement is shown below.

```
constrain collection bookstore with
http://www.25hoursaday.com/books.xsd
```

The above statement retrieves the schema named books.xsd from the URL and validates every document in the collection against it. If all validations are successful then the retrieved document becomes the schema for the collection. This statement fails if the bookstore collection does not exist, there was an error retrieving the schema or if a document in the bookstore collection that does not conform to the schema.

```
ShowStatement ::= 'show constraints on' CollectionClause
```

The show constraints statement displays the schema for the specified collection, if any.

```
DropStatement ::= 'drop constraints on' CollectionClause
```

The drop constraints statement deletes the schema that constrains the contents of a particular collection. Examples of the show constraints and drop constraints statements are shown below.

```
show constraints on collection bookstore
drop constraints on collection bookstore
```

4.4 Queries

The ability to retrieve information from a data source is a fundamental aspect of a data manipulation language. SiXDML constructs enable one to not only query XML

documents and collections but also have a pluggable design that allows for the piping of the results of queries from one query processor to another. Support for embedding other query languages and processors acknowledges the fact that XML repositories vary significantly and some may not store XML natively. Many XML repositories would benefit from the ability to combine queries in the native storage format of the XML repository with a language designed for performing queries of XML documents.

Ideally, an XML query language should support the ability to query all aspects of an XML document's information set as described by the W3C XML information set recommendation [15]. Similarly, support for queries that allow one to query elements and attributes based on their XML namespace [16] information is a desirable feature for an XML data manipulation language. SiXDML will leverage existing W3C XML query language standards to provide the aforementioned functionality.

Combining information from multiple documents analogous to the way that joins are performed in a relational database is also an important feature that an XML data manipulation language should support to provide utmost utility to its users. Query results can be "joined" in various ways; hence defining a fixed semantics for all joins is difficult. In this area, a language such as XQuery that allows one to control the form that query results take is especially useful.

4.4.1 XQuery Syntax for Queries

The querying mechanisms that exist in XQuery are sufficiently powerful enough not to need extension to be useful for SiXDML. Similarly, the ability to specify functions in XQuery will make it possible for XML repositories to couple XQuery syntax with the native syntax of XML repositories to improve the performance of certain queries and perform tasks beyond the purvey of XQuery.

The first example shows an XQuery over an Oracle 9i database that uses an embedded SQL statement within the expression. The query looks in a table called mail_users that contains a single column named mailbox of type XMLType and retrieves all of the users who have an email in their inbox with the string "[xml-dev]" in the subject which indicates that the user is a subscriber to the XML-DEV mailing list .

```
<employees-subscribed-to-xml-dev>{
   for $user in orcl:select("select SYS_XMLGEN(
m.mailbox.extract('/Mailboxes/MailUser')).getclobval()
   from mail_users m")
 where some $email in $user/Inbox/email/subject satisfies
     xf::contains($email/subject/text(), "[xml-dev]")
return <employee>{
   $user/name/text()
}</employee>
}</ employees-subscribed-to-xml-dev >
```

The next example shows an XQuery over an SQL Server 2000 database that uses an embedded SQLXML template. The query uses a mapping schema (not shown here) to create XML views of relational data. An SQLXML template is an XML document containing a combination of XPath and other XML data. When the template is executed the results of each XPath query replaces each of the <xpath-query> elements in the document, which in this case is the list of suppliers in sorted order.

```
let $template := <ROOT>
    <sql:xpath-query mapping-schema="MappingSchema.xml"
     xmlns:sql="urn:schemas-microsoft-com:xml-sql">
        /Supplier
    </sql:xpath-query>
</ROOT>
<Suppliers>
for $supplier in sqlxml:select($template)/ROOT/Supplier
  return $supplier
  sortby (name)
</Suppliers>
```

4.4.2 Alternate Syntax for Queries

```
NamespaceDecl ::= 'namespace' NCName '=' NCName
SelectStatement ::= 'select' XPathExpr 'from' (Collection-
Clause WhereClause? | DocumentPath) ('and transform with'
('xslt' | 'xquery' | 'xpath' | Nmtoken) 'in' (URL | Inlin-
eXml) )?  ('using root =' NmToken)?
WhereClause ::= 'where' XPathExpr
InlineXml ::= '{' ( Comment | PI | CDSect | Element | Char-
Data)+ '}'
```

A select statement executes an XPath query on the specified document or collection and optionally sends the results to another query or stylesheet. The embedded stylesheet or query is either declared inline or retrieved from a particular path for use in further processing of the original query results. The ability to embed other query mechanisms in select statements enables one to overcome the shortcomings of XPath with regards to sophisticated querying facilities as well as use select statements for presentation and not just data retrieval.

If the query is executed against a collection it is returned as XML that conforms to the following schema.

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
   xmlns="xmlns://www.sixdml.org/2002/04/language/"
   elementFormDefault="qualified"
   targetName-
space="xmlns://www.sixdml.org/2002/04/language/">
   <xs:element name="query-results" type="queryResultsType"/>
   <xs:complexType name="queryResultsType">
      <xs:sequence maxOccurs="unbounded">
         <xs:element name="query-result"
type="queryResultType"/>
      </xs:sequence>
      <xs:attribute name="collection-name" type="xs:string"/>
      <xs:attribute name="query" type="xs:string"/>
   </xs:complexType>
   <xs:complexType name="queryResultType" mixed="true">
      <xs:sequence>
         <xs:any minOccurs="0" maxOccurs="unbounded" process-
Contents="skip"/>
      </xs:sequence>
      <xs:attribute name="resource-name" type="xs:string"/>
   </xs:complexType>
</xs:schema>
```

The where clause is used to restrict which documents in the collection are queried by using an XPath expression as a predicate. If a document in the collection returns no

results for the predicate expression in the where clause then the original query is not applied to it and no mention of that document will exist in the XML returned by the query.

In XPath, queries that apply to elements or attributes from a particular namespace must be qualified with a prefix which has been paired with a namespace URI. A namespace declaration defines a namespace prefix and associates it with a namespace URI. The namespace URI must be a valid URI, and may not be an empty string. The namespace declaration is in scope for the rest of the database session or until the prefix is paired with another namespace.

The select statement has two shortcomings that would not exist if XQuery syntax were used for queries. The first shortcoming is that joins, which combine data from multiple sources into a single result, are not supported. Instead one can query an entire collection then pipe the results into a transformation engine such as XQuery or XSLT to perform joins on items within those results. The other shortcoming is that the select statement does not provide a mechanism for executing some queries in the native format of the XML repository as can be done with XQuery functions which could improve performance in some situations. Below are examples of queries using the select statement.

```
namespace bk: = "urn:my-bookstore"
select //bk:title from collection bookstore
```

The above statement selects all the <title> elements in the "urn:my-bookstore" namespace from all the documents in the bookstore collection.

```
select //TITLE[text()='Sandstone'] from auction-
data/auction1.xml and transform with XSLT in
http://www.25hoursaday.com/title.xsl
```

The above statement selects every <TITLE> element from the auction1.xml document in the auctiondata collection that contains the text "SandStone" and passes the results to the specified XSLT stylesheet. If there is more than one <TITLE> element the XSLT processor will complain because the XML passed to it is not well formed. To remedy this, using the root clause can be used to specify a root element for the contents returned by the first part of the select statement. The query thus fails if the XML returned by the XPath query is not well formed. This query fails if there is a problem retrieving the XSLT stylesheet from the URL, and if there are issues with the well formedness or validity of the XSLT stylesheet.

4.5 Data Modification

An important feature of database systems is the ability to insert, update and delete data stored within them. SiXDML provides facilities for modifying aspects of the information set of an XML document that are exposed in the XPath (and Xquery) data model. This includes elements, attributes, text nodes, processing instructions, comments and entire documents.

It should be noted that modifications to a document or collection that is invalid according to the schema for the collection result in an error.

The XQuery syntax used to describe the data modification operations in SiXDML have similar constructs to those in [1] and [4]. The three primary constructs for data modification operations using XQuery syntax are (i) standalone data modification statements (ii) for, let, where or FLW based update expressions and (iii) conditional updates that utilize an if statement. Below is the grammar production showing these three major constructs embedded within the XQuery grammar [2].

```
Query ::= QueryProlog(ExprSequence | UpdateExpr)?
UpdateExprSequence ::= UpdateExpr(Comma UpdateExpr)*
UpdateExpr ::= InsertExpr | DeleteExpr | RenameExpr |
FLWUExpr |     IfUpdateExpr
FLWUExpr ::= (ForClause | LetClause)+ WhereClause? Update-
ExprSequence
IfUpdateExpr ::= If Lpar Expr Rpar Then UpdateExpr Else Up-
dateExpr
```

It should be noted that an XQuery query that performs updates cannot contain expressions that perform queries unless these queries appear as the predicate of a conditional update.

The existence of data modification expressions with multiple update statements can lead to possible conflicts between operations which can lead to undefined results [1], [4]. An example of a conflict that can lead to undefined results is when multiple updates are performed on a single node within a FLWU iteration such as multiple deletions or one or more delete and replace operations. The operations that cause conflicts to occur in SiXDML are the same as those shown in the conflicts table in section 4.3 of [4]. Another thorny issue is the Halloween Problem [18] which appears when an update to a node affects the query expression that determines the source or target of the update expression. An example of a situation that may cause the Halloween Problem is shown below.

```
for $book in /bookstore/book
insert <book /> into /bookstore
```

If in the above statement the update is performed after each iteration and the query reevaluated then the expression loops infinitely. To prevent the Halloween problem from rearing its head and to give clear semantics to the conflict situation, all of the expressions in a data modification statement should be evaluated first before any modifications occur. Thus the following expression, which replaces every occurrence of a particular email address with another.

```
for $b in //book
let $e := $b//author/email[data() = "gte855q@gatech.edu")]
delete $e,
insert <email>kpako@yahoo.com</email> before $e
```

does not result in any conflict.

Data modification operations that result in two text nodes becoming direct siblings result in both nodes being merged into a single text node.

4.5.1 XQuery Syntax for Inserting XML

```
InsertExpr ::= 'insert'
( Expr ( 'into' | 'before' | 'after') Expr ) |
(docFuncCall | Expr 'named' NmToken) 'into' collectionFunc-
Coll)
docFuncCall ::= 'document' '(' Expr ')'
collectionFuncCall ::= 'collection' '(' Expr ')'
```

An insert operation can be used to insert a document into a collection using the into clause or an XML fragment into a document using the into, before and after clauses. Documents can be stored in the repository either by inserting the results of an xf:document function call or a query expression into a collection. When the XML inserted in the collection is the result of an xf:document function call then its name is obtained from its URI. The algorithm for determining this can be implementation specific. On the other hand, XML documents generated from query expressions must be explicitly named before being inserted into a collection. Below is an example of insert statements that utilizes the into clause to add a document to a collection.

```
insert document("http://example.com/books.xml") named
books.xml into     collection("mydb:/bookstore")
```

The above statement fails if the bookstore collection does not exist, there was an error retrieving the document or a if a document named books.xml already exists in the database or if a schema exists for the bookstore collection and books.xml does not conform to the schema.

The into, before and after clauses of the insert statement are used to insert XML fragments into an XML document.

The into clause is used to insert document fragments, processing instructions, comments, text, attributes or any combination thereof as a child of each node that match the target expression. If the target of the insert statement that utilizes the into clause is not one or more element nodes then the statement results in an error. Below are examples of an insert statements that utilize the into clause.

```
for $b in /bookstore/book
insert <rating>unrated</rating> into $b,
   insert attribute in-stock { yes} into $b
```

The before and after clauses are used to insert document fragments, processing instructions, comments, text, or any combination thereof preceding or following each node that match the target expression respectively. If the target of the insert statement that utilizes the before or after clause is not one or more comment, processing-instruction, text, element nodes it results in an error. The operation will also fail if the target node is the document root and an attempt is made to insert anything besides comments or processing instructions. Below is an example of an insert statement that utilizes both the before and after clauses.

```
{-- Insert XSLT stylesheet reference before root node --}
if not(/processing-instruction('xml-stylesheet'))
insert <?xml-stylesheet type="text/xml"
href="bookstore.xsl"?> before /bookstore[1],
insert <!--  beginning of document --> before /bookstore[1],
insert <!--  after of document --> after /bookstore[1]
```

There is some debate as to what should be done when an insert operation utilizes an element or attribute constructor such as in the example below.

```
let $foo
:=<footnotes>document("book.xml")//footnote</footnotes>
insert <!— End of footnote --> into $foo/footnote
```

This case is not explicitly covered in [4] and may cause problems if constructors use copy semantics and do not preserve node identity. Rys's proposal in [1] creates two classes of operations, those that use copy semantics and others that preserve node identity.

In the aforementioned statement, if copy semantics are used then the operation does not perform an update because the $foo/footnote elements are copies of the ones in the document and do not refer to the actual <footnote> elements in the books.xml file. However if the semantics of element constructors were to preserve node identity then the comment would be added as the last child of every <footnote> element in the document.

The XQuery data model specifies that each application of a node constructor create a new node that is identical to itself, and not identical to any other node [17]. To allow for update operations involving constructors to preserve node identity SiXDML adds the concept of views to XQuery. A view is similar to a variable except for the fact that node identity is preserved within the variable. Thus the previous query becomes as follows:

```
let view
$foo := <footnotes>document("book.xml")//footnote</footnotes>
insert <!— End of footnote --> into $foo/footnote
```

4.5.2 Alternate Syntax for Inserting XML

```
InsertStatement ::= 'insert' (In-
sertXmlClause|InsertDocumentClause | InsertAttributeClause)
InsertDocumentClause ::= Url 'named' Nmtoken 'into' Collecti-
onClause  WhereClause?
InsertXmlClause ::= InlineXml (( 'named' Nmtoken 'into' Col-
lectionClause WhereClause? | (( 'before' | 'into' | 'after' )
LocationPath 'in' (CollectionClause WhereClause? |  Document-
Path) ))
InsertAttributeClause ::= 'attribute with name =' QName ',
value=' AttValue (', nsuri =' NSName)? 'into' LocationPath
'in' (CollectionClause WhereClause? | DocumentPath)
```

The semantics of the insert statement with the into, before, or after clauses are similar to that described in section 4..5.1 except for the introduction of the where clause. The where clause is used to selectively choose which documents should be updated by the query. Below are some examples of using the insert statement.

```
insert http://www.example.com/books.xml named books.xml into
collection bookstore
insert{
<!-- The following element is a book -->
} before //book in bookstore/books.xml
insert attribute with name="restock", value="yes" into /root
in collection bookstore where count(//book) < 5
```

4.5.3 XQuery Syntax for Deleting XML

```
DeleteExpr ::= 'delete' Expr
```

The delete statement removes all nodes that match the target expression. The statement returns an error if the target expression contains a construction or deletes nodes that prevent the document from being well formed. Deletion of the root node of a document removes it from the XML repository. An example of the delete statement is shown below.

```
{-- Remove all comments in the document --}
delete //comment( )
```

4.5.4 Alternate Syntax for Deleting XML

```
DeleteStatement ::= 'delete' LocationPath 'from' (Collection-
Clause WhereClause? | DocumentPath)
```

The semantics of the delete statement in the alternate syntax are similar to that described in section 4..5.3 except for the introduction of the where clause. The where clause is used to selectively choose which documents should be modified by the query. Below are some examples of using the delete statement.

```
delete //comment( ) from bookstore/books.xml
delete /root/@restock from collection bookstore where
count(//book) > 10
```

4.5.5 XQuery Syntax for Replacing XML

```
ReplaceExpr ::= 'replace' Expr 'with' Expr
```

The semantics of the replace statement are that each node in the first expression is replaced by all the nodes in the secondary expression of the statement. However, a replace is not equivalent to a delete followed by an insert because there exist situations where a delete followed by an insert might fail (because the document would be temporarily invalid) but a replace would succeed. A FLWUExpr containing a delete and an insert may be considered analogous to a replace due to the fact that a FLWUExpr is supposed to be atomic.

The operation fails if the replacement results in a document that is invalid. For example, replacing attribute with nodes other than attribute nodes or replacing the root element with multiple elements will result in an error.

The following replace statement:

```
replace //score with <rating>unrated</rating>
```

is equivalent to the following insert and delete operation:

```
let $s := //score
insert <rating>unrated</unrated> before $s
delete $s
```

4.5.6 Alternate Syntax for Replacing XML

```
ReplaceStatement ::= 'replace' LocationPath 'with' InlineXml
'in' (CollectionClause WhereClause?| DocumentPath)
```

The semantics of the replace statement in the alternate syntax is similar to that described in section 4..5.5 except for the introduction of the where clause. The where clause is used to selectively choose which documents should be modified by the query. Below is an example of using the replace statement.

```
replace //price with {
<price>Unavailable</price>
<!--Prices are being reduced by 20% or 25% if > $10 -->
<rating>Unrated</rating>
} in bookstore/books.xml
```

4.5.7 XQuery Syntax Renaming Elements and Attributes

```
RenameExpr ::= 'rename' Expr 'to' Expr
```

The rename statement can be used to rename attribute and elements nodes Attempts to rename other types of nodes will result in an error.

The first expression is not allowed to contain a construction and returns an error if this is not the case. The second expression should result in a single XML qualified name and returns an error if this is not the case. Below is an example of a rename statement.

```
rename /root-element to "start-node"
```

Renaming elements in documents from a constrained collection will typically result in failure because most XML schema technologies consider names to be significant to an element's type. However this may be acceptable if the elements are members of a wildcard [14] specified in the collection's schema.

4.5.8 Alternate Syntax for Renaming Elements and Attributes

```
RenameStatement ::= 'rename' LocationPath 'to' QName 'in'
(CollectionClause WhereClause? | DocumentPath)
```

The semantics of the rename statement in the alternate syntax is similar to that described in section 4..5.7 except for the introduction of the where clause. The where clause is used to selectively choose which documents should be modified by the query. Below is an example of using the rename statement.

```
namespace bk = "urn:my-books"
rename //bk:magazine to "bk:journal" in collection
book-repository where /bk:library
```

which renames magazine elements from the "urn:my-books" namespace to journal elements in documents that have a library root element.

5 Conclusion

The language presented in this paper tackles some of the features that users of hierarchical and relational database systems take for granted but currently do not exist in the XML database world. Although a large body of research has been done on creating query and transformation languages for XML, little headway has been made in creating a language that hits the 80/20 point in satisfying the requirements of XML database users.

SiXDML combines data definition, data manipulation and database management in a single package with the goal of simplifying the disparate mechanisms that currently exist in the traditional XML database systems for performing the aforementioned tasks.

It is our belief that SiXDML or a language will be instrumental in increasing the proliferation of XML databases by finally providing a simple and unified means to work with XML in a database context.

The SiXDML prototype is available at:

`http://www.25hoursaday.com/sixdml/demo.html`

Acknowledgements

The following people contributed to this paper with their insight and comments: Michael Rys, Michael Brundage, Mark Fussell, Kimbro Staken, Tom Bradford, Mike Champion, Ronald Bourrett and Sanjay Bhatia.

References

[1] Rys, M.: Proposal for an XML Data Modification Language. Microsoft Report. 2002.

[2] Boag, S., Chamberlin, D., Fernandez, M.F., Florescu, D., Robie, J., Siméon, J., Stefanescu, M.: XQuery 1.0: An XML Query Language (Working Draft). 20 December 2001. http://www.w3.org/TR/2001/WD-xquery-20011220.

[3] Clark, J., DeRose, S.: XPath: XML Path Language (Version 1.0). 16 November 1999. http://www.w3.org/TR/1999/REC-xpath-19991116.

[4] Lehti, P.: Design and Implementation of a Data Manipulation Processor for an XML Query Language. 2001. http://www.lehti.de/beruf/diplomarbeit.pdf.

[5] Milo, T., Suciu, D., Vianu, V.: Typechecking For XML Transformers. In: Proceedings of the 19th ACM Symposium on Principles of Database Systems, pp. 11-22. ACM Press, 2000.

[6] Alon, N., Milo, T., Neven, F., Suciu, D., Vianu, V.: XML with Data Values: Typechecking Revisited. In: Proceedings of the 20th ACM Symposium on Principles of Database Systems, 2001.

[7] Staken, K.: XML:DB Database API (Working Draft). 20 September 2001. http://www.xmldb.org/xapi/xapi-draft.html.
[8] Laux, A.: XUpdate - XML Update Language. 14 September 2000. http://www.xmldb.org/xupdate/xupdate-wd.html.
[9] Kapp, D., Leben, J.F.: IMS Programming Techniques: A Guide To Using DL/I. New York: Van Nostrand Reinhold. 1986.
[10] Date, C.J.: An Introduction to Database Systems. 7th ed. Addison-Wesley. 2000.
[11] Elmasri, R., Navathe, S.B.: Fundamentals of Database Systems. 3rd edition, Addison-Wesley. 2000.
[12] Chamberlain, D., Fankhauser, P., Marchiori, M., Robie, J.: XQuery Requirements. 15 February 2002. http://www.w3.org/TR/2001/WD-xmlquery-req-20010215.
[13] Bray, T., Paoli, J., Sperberg-McQueen, C.M., Maler, E.: Extensible Markup Language (XML) 1.0 Second Edition. 6 October 2000. http://www.w3.org/TR/2000/REC-xml-20001006.
[14] Thompson, H.S., Beech, D., Maloney, M., Mendelsohn, N.: XML Schema Part 1: Structures. 2 May 2001. http://www.w3.org/TR/2001/REC-xmlschema-1-20010502/.
[15] Cowan, J., Tobin, R.: XML Information Set. 24 October 2001. http://www.w3.org/TR/2001/REC-xml-infoset-20011024.
[16] Bray, T., Hollander, A., Layman, A.: Namespaces in XML. 14 January 1999. http://www.w3.org/TR/1999/REC-xml-names-19990114.
[17] Fernández, M., Marsh, J., Nagy, M.: XQuery 1.0 and XPath 2.0 Data Model. 20 December 2001. http://www.w3.org/TR/2001/WD-query-datamodel-20011220/.
[18] Gray, J. (ed.): The Benchmark Handbook. Morgan Kauffmann. 1993.
[19] Malhotra, A., Rys, M., Robie, J.: XML Syntax for XQuery 1.0 (XQueryX). 7 June 2001. http://www.w3.org/TR/2001/WD-xqueryx-20010607.
[20] Thompson, H.S., Frankston, C.: XML- Data reduced (version 0.21). 3 July 1998. http://www.ltg.ed.ac.uk/~ht/XMLData-Reduced.htm.
[21] Clark, J., Makoto, M.: Relax NG Specification. 3 December 2001. http://www.oasis-open.org/committees/relax-ng/spec-20011203.html.
[22] Malhotra, A., Melton, J., Robie, J., Walsh, N.: XQuery 1.0 and XPath 2.0 Functions and Operators. 7 Mar 2002. http://www.w3.org/TR/xquery-operators/.
[23] Biron, P.V., Malhotra, A.: XML Schema Part 2: Datatypes. http://www.w3.org/TR/2001/REC-xmlschema-2-20010502/.

Relevance Ranking Tuning for Similarity Queries on XML Data

Paolo Ciaccia and Wilma Penzo

DEIS - CSITE-CNR
University of Bologna, Italy
{pciaccia,wpenzo}@deis.unibo.it

Abstract. Similarity query techniques integrating semantics and structure of XML data have been recently investigated. Mostly, query relaxations are not fully exploited to retrieve data that approximate query conditions on data organization. In this paper we present a method to widen the spectrum of relevant data, and we define a measure to *tune* the ranking of results, so that also information on *relevance quality* of data retrieved can be inferred from scores.

1 Introduction and Motivation

Several researchers have recently profused their efforts in the integration of semantics and structure for querying XML data [5, 7, 13]. Because of the heterogeneity of large digital libraries, and in absence of knowledge of data organization, Boolean query conditions often lead to empty results. Thus, relaxation on query requirements has been acknowledged as a "must". As to data content, some proposals suggest the use of vague predicates to formulate queries [7, 13]. These assume thesauri, ontologies, and semantic networks to find out the relevance of terms inside documents. As to data organization, relaxations of structural conditions only allow for "larger results", where *all* hierarchical dependencies between data are (somehow) preserved [11]. This is the case of path expressions containing wildcards. The basic idea is that results are relevant no matter how "sparse" they are. This flattens the relevance of the retrieved data. In fact, although all required information is found, it possibly appears in different contexts, and this is supposed to affect relevance. Further, a total match approach is restrictive, since it limits the set of relevant results. Actually, most of existing works do not cope with the problem of providing *structurally incomplete results*, i.e. results that only partially satisfy query requirements on text organization.

Let us consider, for instance, a user looking for stores selling CD's authored by a singer whose lastname is "John"[1], and containing songs with "love" in the title. Among the documents shown in Fig. 1, only `Doc1` fully satisfies query requirements. This is because `Doc1` is the only document that presents a "lastname" tag, thus making condition on "John" checkable. But, it is evident that also documents `Doc2` is relevant to the query, and should be returned.

[1] Suppose the user does not remember the firstname "Elton" and, since "John" is a very common name, she wants to specify that "John" must be a lastname.

S. Bressan et al. (Eds.): EEXTT and DIWeb 2002, LNCS 2590, pp. 22–34, 2003.

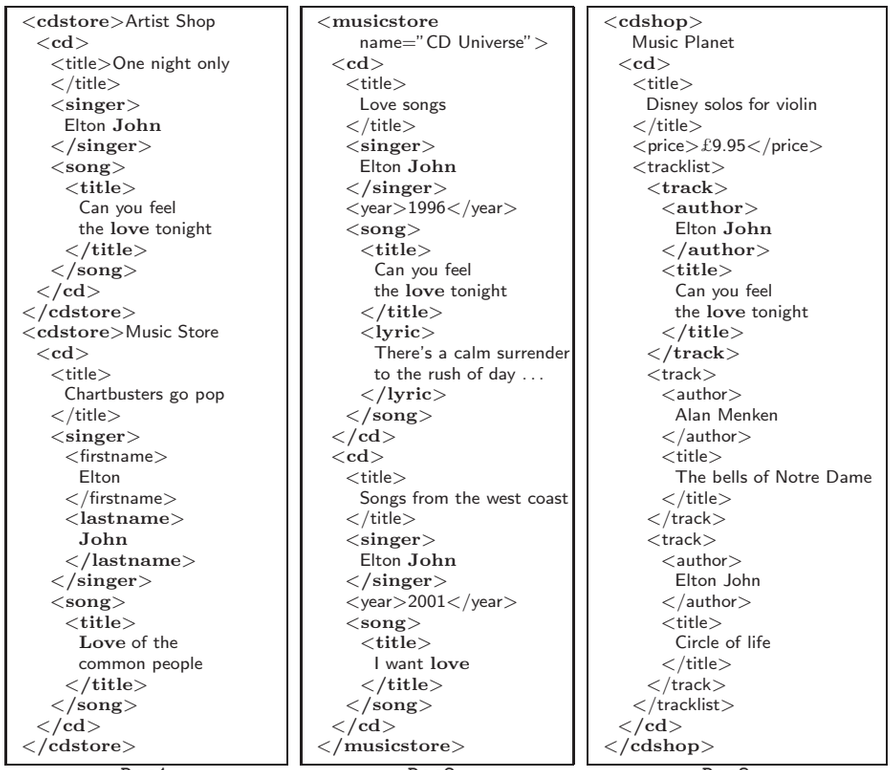


Fig. 1. Sample XML documents

The above-cited approaches do not capture this approximation, except for ApproXQL [12] and [5]. In these proposals, however, both `Doc1` and `Doc2` would be evaluated with the same score, although `Doc2` contains two relevant CD's rather than only one as in `Doc1`. Further, also `Doc3` is expected to be returned as relevant, and it is not. This is because data is organized in a slight different way: Elton John appears as a song author rather than as the CD author, as specified by the query. This structure disagreement does not allow to recognize `Doc3` as relevant. This points out that evaluation of structure conditions should be made more flexible. Basically, a finer *tuning* of relevance scores should be provided: partial matches on structure, as well as approximations, actually are supposed to influence relevance, but should not be the cause of data discarding. Then, documents that present more occurrences of query requirements are expected to score higher, and this should be captured by an effective measure of relevance. Current methods usually produce absolute ranking values that, individually, do not provide information on the query rate satisfied by an answer. For instance, it is not possible to realize if, given a query, a document is assigned a low

score because of an approximate yet complete match, or because of a partial yet exact match of query requirements. Since the output of a query is usually large, additional information that justifies scores would lighten the user from the burden of discovering which solutions come up better to her expectations. Thus, in general, an *effective* relevance ranking method is expected to provide information to infer the overall *quality* of data retrieved.

In this paper we provide a method to find the *approximate embeddings* of a user query in XML document collections. Our proposal captures the relaxations described above, and provides a measure that, besides ranking, also specifies *quality* of results. The outline is as follows: In Section 2 we introduce a basic query language and a tree representation for queries and documents. In Section 3 we start from the unordered tree inclusion problem [9] to relate query and data trees through *embeddings*; this is extended in two directions: 1) to capture also partial matching on query structure, and 2) to assign a score to the retrieved embeddings. To this end, we define the $SATES$ (*Scored Approximate Tree Embedding Set*) function, to retrieve and score embeddings. We also show how some "critical" queries are effectively managed by our method. Our relevance ranking measure is then presented. Then, we compare our method with other approaches in Section 4 and, finally, in Section 5 we conclude and discuss future directions we intend to follow.

2 XML Query Language and Trees

XML *(eXtensible Markup Language)* is a markup language for defining content and organization of text documents. For the sake of simplicity, we do not require validity for XML documents, in that we do not assume the presence of DTDs. We also do not consider further features (e.g. IDREFs, namespaces) provided by the XML specification [2].

As a starting point, we consider a subset of the XQuery1.0 grammar [4], that we call $XQuery^{-}$. Queries are path expressions, with predicates restricted to equality on attribute and element values. An $XQuery^{-}$ expression has the form:

```
[<sep>](<StepExpr><sep>)*<PrimaryExpr>"["<Expr>"]"
```

with the `<sep>` hierarchical separator having possible values "/" and "//" to denote parent-child and ancestor-descendant relationships, respectively; the expression `(<StepExpr><sep>)*<PrimaryExpr>` denotes a path in a document where conditions are expected to be checked in; `<PrimaryExpr>` indicates the query output; conditions are expressed between square brackets, as a boolean combination of predicates. The following is an example of complex query:

Example 1 *Retrieve data about musical CDs on sale at stores in Manhattan, where author's lastname is "John", being produced in "1996", and containing songs' lyrics:*

```
cdstore/cd[author/lastname=‘‘John’’ and ../address/* =
  ‘‘Manhattan’’ and @year=‘‘1996’’ and .//song/lyric]
```

$XQuery^-$ expressions are mapped to trees in a natural way [3]. Trees are made of typed labelled nodes and (possibly labelled) edges. Nodes may have the following types: `element`, `attribute` and `content`. A `content` leaf expresses the value its parent `element/attribute` node is required to assume. An `element/attribute` leaf specifies a condition on the presence of that specific attribute/element in the given context. Non-leaf nodes denote the context where information is expected to be found in. Finally, each edge connecting two nodes either model simple parent-child relationship, or it may be labelled with the "$*$" symbol, to denote a path of positive length, thus expressing an ancestor-descendant relationship.

We briefly recall some basic definitions on trees, and we introduce some additional functions. A *tree* t is a pair (N, E), with N finite set of typed labelled nodes, and E binary relation on N. Hierarchical relationships on a tree t, are defined as: $\forall n_1, n_2 \in N$, 1) $parent(n_1, n_2) \iff (n_1, n_2) \in E$; 2) $ancestor(n_1, n_2) \iff (parent(n_1, n_2) \vee \exists n \in N$ such that $parent(n_1, n) \wedge ancestor(n, n_2))$.

$\exists! r \in N$ such that: 1) $\nexists n \in N$ such that $parent(n, r)$; 2) $\forall n \in N, n \neq r$, $ancestor(r, n)$. r is called the *root* of the tree. $\forall n_2 \in N$, $\exists! n_1 \in N$ such that $parent(n_1, n_2)$. If $ancestor(n_1, n_2)$ holds, the sequence of edges that connect n_1 with n_2 is called a *path* from n_1 to n_2.

Let $\mathcal{T}$ be the set of all typed and labelled trees: $\forall t \in \mathcal{T}$, *root(t)* and *nodes(t)* return the root and the set of all nodes of the tree t, respectively. Given a tree $t = (N, E)$, $\forall n \in N$, *label(n)* and *type(n)* return the label $l \in \mathcal{L}$, with $\mathcal{L}$ set of labels, and the type $k \in \mathcal{K}$, with $\mathcal{K}$ set of node types, for a node n, respectively; *children(n)* returns the set of all children of node n; *leaf(n)* $\iff$ *children(n)* $= \emptyset$; *support(n)* is the subtree of t rooted at n, called the *support* of n in t.

We provide a tree representation also for documents so as to reason on query and document similarity by means of tree comparison. We want to determine how much a document tree satisfies semantics and structural constraints provided by a query tree. Let $\mathcal{D}$ be the set of well-formed XML documents.

Definition 1 (From XML Docs to trees) Given a document $d \in \mathcal{D}$, its corresponding tree $t_d \in \mathcal{T}$, is such that each element, attribute, and content data in d is mapped to a node n_e, n_a, and n_c, in t_d, respectively, and:

1. $type(n_e) =$ `element`, $type(n_a) =$ `attribute`, $type(n_c) =$ `content`;
2. labels of nodes are element's and attribute's names, and content data values;
3. each nesting level of element/attribute/content data in d resolve to a *parent-child* relationship between corresponding nodes in t_d.

Let $\mathcal{T}_Q \subset \mathcal{T}$ and $\mathcal{T}_D \subset \mathcal{T}$ be the sets of query and data trees, respectively.

3 The Tree Embedding Problem

Satisfying a query q on a document d may lead to different results, depending on how much we are willing to relax constraints on query semantics and requirement

dependencies. In our tree view, when a query has to be completely satisfied, we look for an *embedding* of a query tree t_q in a document tree t_d. This means that all query nodes must have a corresponding matching node in the document tree, and each parent-child relationship should be guaranteed at least by an ancestor-descendant one in the data tree [9]. In many cases these conditions are too restrictive, since data may not correspond completely and exactly to query requirements.
Our work basically loosens the strictness of this approach, that often leads to empty results. The key aspects of our proposal are:

1. The relaxation on the concept of *total* embedding of t_q in t_d, in that we admit *partial* structural match of the query tree; further, exact match is relaxed to consider semantic similarity between nodes;
2. Approximate results, that are *ranked* with respect to the *cohesion* of retrieved data, to the relaxation of semantic and structural constraints, and to the *coverage rate* of the query. Our ranking function takes into account the overall query satisfaction, in that a score provides a ranking value but also a *quality measure* of results;
3. A set-oriented approach to results, that depending on different relaxations on requirements, identifies possible alternatives that the user may be interested in. This strengthens the relevance of data presenting multiple occurrences of query patterns.

Point 1) leads to the introduction of our interpretation of *approximate tree embedding*.

Definition 2 (Approximate Tree Embedding) Given a query tree $t_q \in \mathcal{T}_Q$ and a document tree $t_d \in \mathcal{T}_D$, an *approximate tree embedding* of t_q in t_d is a *partial* injective function $\tilde{e}[t_q, t_d] : nodes(t_q) \nrightarrow nodes(t_d)$ such that $\forall q_i, q_j$ in the domain of $\tilde{e}$ ($dom(\tilde{e})$):

1. $sim(label(q_i), label(\tilde{e}(q_i))) > 0$, with *sim* a similarity operator that returns a score in [0,1] stating the semantic similarity between the two given labels
2. $parent(q_i, q_j) \Rightarrow ancestor(\tilde{e}(q_i), \tilde{e}(q_j))$

Let $\mathcal{E}$ be the set of approximate tree embeddings.

As to the point 2), in order to specify the ranking of results, embeddings are to be assigned a *score*, according to a *relevance ranking function* ρ.

Definition 3 (Relevance Ranking Function) We denote with ρ a *ranking function* that, given a triple $(t_q, t_d, \tilde{e}[t_q, t_d])$, returns a tuple:

$$\Sigma = (\sigma, \gamma_1, \gamma_2, \gamma_3, \gamma_4, \gamma_5)$$

of *scores* with values in $\mathcal{S} = [0, 1]$ such that, with respect to the approximate embedding $\tilde{e}$ of the query expressed by t_q in the document expressed by t_d:

- γ_1 indicates how much $\tilde{e}$ is *semantically complete* with respect to the query;
- γ_2 denotes *semantic correctness* of $\tilde{e}$, in that it states how well the embedding satisfies semantic requirements;
- γ_3 represents the *structural completeness* of $\tilde{e}$ with respect to the given query; it denotes the structural *coverage* of the query;
- γ_4 expresses *structural correctness* of $\tilde{e}$, in that it is a measure of how well constraints on structure are respected in the embedding;
- γ_5 specifies *cohesion* of $\tilde{e}$, by providing the grade of fragmentation of the retrieved embedding;
- σ states the *overall similarity score* of the embedding, and it is obtained as a combination of $\gamma_1, \gamma_2, \gamma_3, \gamma_4$, and γ_5.

Formally: $\rho : \mathcal{T}_Q \times \mathcal{T}_D \times \mathcal{E} \rightarrow S^6$

Now we are ready to introduce a *scored approximate tree embedding*. For the sake of simplicity, we start considering embeddings scored with respect to the overall scoring value σ. Complete relevance score computation is detailed in Section 3.1.

Definition 4 (Scored Approximate Tree Embedding) A *scored approximate tree embedding* $\tilde{e}_s$ is an approximate tree embedding extended with a *score* in $\mathcal{S}$. Formally: $\tilde{e}_s : \mathcal{S} \times \mathcal{E}$.

The similarity function we propose for retrieval and scoring of embeddings of a query tree in a document tree, is given by the $SATES$ (*Scored Approximate Tree Embedding Set*) *function*, shown in Fig. 2.

Definition 5 ($SATES$ Function) We define the *Scored Approximate Tree Embedding Set Function* as:

$$SATES : \mathcal{T}_Q \times \mathcal{T}_D \rightarrow 2^{\mathcal{S} \times \mathcal{E}}$$

$\forall t_q \in \mathcal{T}_Q, \forall t_d \in \mathcal{T}_D$, $SATES(t_q, t_d)$ returns a set of scored approximate tree embeddings for t_q in t_d. This captures the possibility of having more than one embedding between a query tree and a document tree.

Intuitively, the $SATES$ function states "how well" a data tree t_d fits a query tree t_q, also taking care of multiple fittings. In order to determine the scored embeddings, the function follows a recursive definition and examines different cases for t_q and t_d:

Both Leaves. Depending on the semantic similarity of node labels, the result set is made of: 1) one scored embedding that relates the two nodes; 2) the emptyset if no similarity is found for labels. If an embedding is returned, since structural similarity between leaves can be considered "perfect", semantic similarity assumes the key role of determining the final score for the embedding.

Query Leaf and Data Tree. If the semantic similarity between roots' labels is positive, an embedding is found and its score is determined as in the previous

$\forall t_q \in \mathcal{T}_Q, t_d \in \mathcal{T}_D, SATES(t_q, t_d)$ is defined as:

case $leaf(root(t_q)) \wedge leaf(root(t_d))$:
  if $sim(t_q, t_d) > 0$
    $SATES(t_q, t_d) = \{[(s(t_q, t_d), \{(root(t_q), root(t_d))\}]\}$
  else $SATES(t_q, t_d) = \emptyset$

case $leaf(root(t_q)) \wedge \neg leaf(root(t_d))$:
  if $sim(t_q, t_d) > 0$
    $SATES(t_q, t_d) = \{[(s(t_q, t_d), \{(root(t_q), root(t_d)\})]\}$
  else $SATES(t_q, t_d) = \bigcup_{c \,\in\, children(t_d)} \ominus_d \, (SATES(t_q, support(c)))$

case $\neg leaf(root(t_q)) \wedge leaf(root(t_d))$:
  if $sim(t_q, t_d) > 0$
    $SATES(t_q, t_d) = \{[(s(t_q, t_d), \{(root(t_q), root(t_d)\})]\}$
  else $SATES(t_q, t_d) = \bigcup_{c \,\in\, children(t_q)} \ominus_q \, (SATES(support(c), t_d))$

case $\neg leaf(root(t_q)) \wedge \neg leaf(root(t_d))$:
  if $sim(t_q, t_d) > 0$ $SATES(t_q, t_d) =$

$$\bigcup_{\mathcal{M} \in \mathcal{M}_{t_q}^{t_d}} \; \bigcup_{\substack{(t_i^k, t_j^k) \,\in\, \mathcal{M} \\ (s_{l_k}^k, m_{l_k}^k) \,\in\, SATES(t_i^k, t_j^k) \\ l_k \,\in\, [1..|SATES(t_i^k, t_j^k)|]}} [\otimes(s(t_q, t_d), s_{l_1}^1, \ldots, s_{l_{|\mathcal{M}|}}^{|\mathcal{M}|}), \{(root(t_q), root(t_d))\} \cup m_{l_1}^1 \cup \ldots \cup m_{l_{|\mathcal{M}|}}^{|\mathcal{M}|}]$$

  else $SATES(t_q, t_d) = \bigcup(\bigcup_1, \bigcup_2, \bigcup_3)$

$$\text{where } \bigcup_1 = \bigcup_{\mathcal{M} \in \mathcal{M}_{t_q}^{t_d}} \ominus_q \circ \ominus_d \bigcup_{\substack{(t_i^k, t_j^k) \,\in\, \mathcal{M} \\ (s_{l_k}^k, m_{l_k}^k) \,\in\, SATES(t_i^k, t_j^k) \\ l_k \,\in\, [1..|SATES(t_i^k, t_j^k)|]}} [\otimes(s_{l_1}^1, \ldots, s_{l_{|\mathcal{M}|}}^{|\mathcal{M}|}), m_{l_1}^1 \cup \ldots \cup m_{l_{|\mathcal{M}|}}^{|\mathcal{M}|}]$$

$$\bigcup_2 = \bigcup_{c \,\in\, children(t_d)} \ominus_d \, (SATES(t_q, c))$$

$$\bigcup_3 = \bigcup_{c \,\in\, children(t_q)} \ominus_q \, (SATES(c, t_d))$$

Fig. 2. The $SATES$ Function

case. Here note that, even if the structure of the two trees is different (one of them is indeed a leaf), the embedding's score should not be influenced from this. In fact, we can evidently conclude that the *structural coverage* is complete. In case of null semantic similarity, t_d's children are entrusted to determine some possible embeddings for the leaf t_q. Then, the scores of the resulting embeddings are *lowered* through the $\ominus_d$ function,[2] so that the skip of the root of t_d, that did not match with the query node, is taken into account in the computation of the structural similarity between t_q and t_d. Even if we can say that t_d *covers* t_q, indeed it provides a more generic context, that does not "exactly" satisfy

[2] Notation details are shown later in this section.

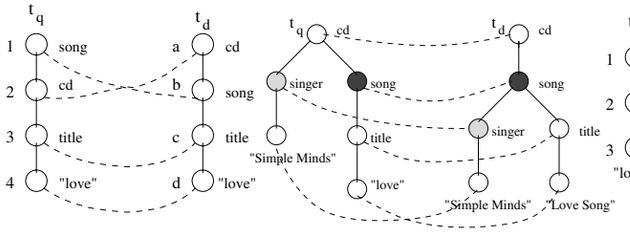


Fig. 3. Swap

Fig. 4. Unbalance

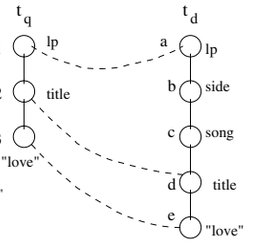


Fig. 5. Low cohesion

structural requirements (and consequently also semantics). Score lowering can then be considered appropriate.

Query Tree and Data Leaf. This is the case that concludes recursion in the next step. Its role is made evident by the following case.

Query Tree and Data Tree. This is the general case that usually starts a $SATES$ call. The result set is determined in a recursive fashion. Basically, two cases may occur: either 1) semantic similarity of roots' labels exists, or 2) no similarity is found. The former is the simplest case. The idea is that, since roots are semantically related, this means that the current *data context* is promising for the examination of remaining query conditions. Thus, query root's children get involved in the recursive computation of embeddings. Matchings are considered in the bipartite graph[3] between query root's children and data root's children.

In case 2) of null label similarity, the final embeddings are computed from the union of three quantities, that, for simplicity, we call $\bigcup_1$, $\bigcup_2$, and $\bigcup_3$. Basically, the final embeddings must consider the unsuccessful match of the roots. This implies the final scores to be properly lowered. Then, two strategies can be followed: either looking for satisfying the remaining query conditions in the current context, or changing the context. Consider Fig. 3. Assume semantic similarity between roots' labels "song" and "cd" is 0. Before asserting that certainly no embeddings exist, we proceed in two directions: 1) We change the context of our search, trying to satisfy the query in a more specific domain: thus we move our target to the child(ren) of t_d's root ($\bigcup_2$ computation); 2) we give up looking for a complete match of the query, and we try to discover if the current context satisfies at least the remaining conditions ($\bigcup_3$ computation). Of course, in both cases result scores are properly lowered to take into account the presence of unmatched nodes. This "crossed comparison" may point out possible "swaps" between query and data nodes. In fact, in our example, two embeddings are re-

[3] Consider a graph $G = (V, E)$. G is *bipartite* if there is a partition $V = A \cup B$ of the nodes of G such that every edge of G has one endpoint in A and one endpoint in B. A *Matching* is a subset of the edges no two of which share the same endpoint. In our case $A = children(root(t_q))$ and $B = children(root(t_d))$.

trieved: {(1,b),(3,c),(4,d)} and {(2,a),(3,c),(4,d)}. Even if they are two separate solutions, it is worth noticing that the union of the two embeddings provides a full coverage for the query tree. Thus, a more complete solution could be derived, albeit including score penalization for the swap. Further, $\bigcup_2$ computation captures also structural dissimilarites like that shown in Fig. 4 where a possible embedding is dashed. Note that, "song" and "singer" nodes are siblings in t_q and parent-child in t_d. This should penalize the final score of the embedding, and it is captured by the $\ominus_d$ function when computing the embedding of *support*("singer") in *support*("song").[4]

Further, when comparing subtrees as an effect of recursive calls of the $SATES$ function, the event of unsuccessful match for the document (current) root may be interpreted as a sign of low cohesion of the global result. As an example, let consider the trees of Fig. 5. The evaluation of $SATES(t_q, t_d)$ leads to the embedding {(1,a),(2,d),(3,e)}. In this case, although the query tree is totally embedded in the document tree, cohesion of retrieved data is rather low. This is captured by the double application of the $\ominus_d$ function, because of the two recursive steps with unsuccessful match of (2,b) and (2,c). Note that the $SATES$ function is not symmetric. Consider trees in Fig. 5 again. Finding the "inverse" approximate embedding of t_d in t_q would result in a partial satisfaction of query requirements since "side" and "song" elements do not have a correspondance in t_q. This means that the $\ominus_q$ lowering function "weighs" differently (much more indeed) from $\ominus_d$. Thus, the retrieved approximate embeddings would be scored differently. This captures the intuition that priority is on satisfaction of query requirements, and then on cohesion of results.

***SATES* Formal Notation** To reduce notational complexity in the definition of the $SATES$ function provided in Fig. 2, we used some abbreviations: $sim(t_q, t_d)$ stands for $sim(label(root(t_q)), label(root(t_d)))$, and $s(t_q, t_d)$ is the score obtained by the ρ function with reference to the base case of embedding $root(t_q)$ in $root(t_d)$. Then, $\mathcal{M}_{t_q}^{t_d}$ is the *Bipartite Graph Matching* between the two sets $children(root(t_q))$ and $children(root(t_d))$. When clearly defined in the context, we use $\mathcal{M}$ in place of $\mathcal{M}_{t_q}^{t_d}$. The $\ominus_q$, $\ominus_d$, and $\otimes$ functions are defined as follows.

Definition 6 (Lowering Functions) The $\ominus_q$ and $\ominus_d$ functions change the scores of a given a set of scored approximate tree embeddings, according to a lowering factor. New scores capture the unsuccessful match of a query node and a data node, respectively. Formally:
$\ominus_q$: $2^{\mathcal{S}\times\mathcal{E}} \rightarrow 2^{\mathcal{S}\times\mathcal{E}}$ such that $\forall \tilde{e}_s = (s, \tilde{e}) \in dom(\ominus_q)\ \ominus_q(\tilde{e}_s) = (s', \tilde{e}) \wedge s' \leq s$.
$\ominus_d$ is defined similarly.

Definition 7 (Combine Function) The $\otimes$ function generates a new score from a set of n given scores in $\mathcal{S}$. Formally: $\otimes : 2^{\mathcal{S}} \rightarrow \mathcal{S}$.

[4] Note that labels of trees' roots match.

3.1 Relevance Computation

An *effective* relevance ranking method is expected to provide information to infer *quality* of data retrieved. Most approaches rank results according to scores that depend on the satisfyiability of matching query conditions [7, 13]. Others use cost functions to combine the costs of relaxations required to (also partially) satisfy the query [5, 12]. In this case, combination (usually sum) produces *absolute* ranking values that, individually, do not provide information on how much of the query is satisfied by an answer. Assume to compare results coming from two different queries q_1 and q_2. According to these scoring functions [5, 12], it is possible that one document that satisfies 1 condition (out of 2) of q_1 is assigned the same score of a document that satisfies 9 conditions (out of 10) of the more complex query q_2. Although results are incomparable, one would expect documents (exactly) satisfying a high percentual of conditions to score higher than documents (exactly) satisfying a lower rate.

Thus, besides information on *correctness* of results, a measure of *completeness* is desirable. As to XML documents, this information is somehow made more complex by the presence of structure inside documents. This implies that, in addition to a measure of semantic satisfaction, as in the keyword case, some knowledge on completeness is supposed to provide also information on the matching rate of query structure. Also, *cohesion* of data retrieved is another important feature to be considered. Cohesion provides a measure of how "sparse" the required information is. Apart from queries where the user explicitly specifies not to take care of the depth where information may be found in, this information is an important element to be considered when ranking data. As to this property, existing approaches [7, 13] do not treat this information. As a consequence, *quality* of results can be considered an overall measure of all these features.

We provide a scoring method that takes into account semantic and structural completeness and correctness of results, as well as cohesion of relevant data retrieved. We start modeling a set of properties for each embedding $\tilde{e}$. Property values are normalized in the interval $[0, 1]$, where values close to 1 denote high satisfaction. Then, we discuss how these measures can be combined to obtain an overall score for an embedding. Properties are:

Semantic Completeness. It is a measure of how much the embedding is semantically complete with respect to the given query. It is computed as the ratio between the number of query nodes in the embedding, $n_q^{\tilde{e}}$, and the total number of query nodes, n_q:

$$\gamma_1 = \frac{n_q^{\tilde{e}}}{n_q}$$

Semantic Correctness. It states how well the embedding satisfies semantic requirements. It represents the overall semantic similarity captured by the nodes in the embedding. This is computed as a combination $\wedge$ (for instance, product) of label similarities of matching nodes, possibly lowered by type mismatchings (attribute vs. element nodes):

$$\gamma_2 = \bigwedge_{q_i \in dom(\mathcal{E})} sim(label(q_i), label(\tilde{e}(q_i)))$$

Structural Completeness. It represents the *structural coverage* of the query tree. It is computed as the ratio between: 1) the number of node pairs in the image of the embedding, $hp_q^{\tilde{e}}$, that satisfy the same hierarchical[5] relationship of the query node pairs which are related to, and 2) the total number of hierarchy-related pairs in the query tree (hp_q):

$$\gamma_3 = \frac{hp_q^{\tilde{e}}}{hp_q}$$

Structural Correctness. It is a measure of how many nodes respect structural constraints. It is computed as the complement of the ratio between the number of structural penalties p (i.e. swaps and unbalances) and the total number of hierarchy-related pairs in the data tree that also appear in the embedding:

$$\gamma_4 = 1 - \frac{p}{hp_d^{\tilde{e}}}$$

Cohesion of Results. It represents the grade of fragmentation of the resulting embedding. It is computed as the complement of the ratio between the number of intermediate data nodes $in_d^{\tilde{e}}$ and the total number of data nodes in the embedding, also including the intermediate ones $n_d^{\tilde{e}}$:

$$\gamma_5 = 1 - \frac{in_d^{\tilde{e}}}{n_d^{\tilde{e}}}$$

These properties can be naturally partitioned in two sets: properties related to semantics, and properties concerning structure. A combination of them indicates a global measure, that summarizes the semantic and structural characteristics of the retrieved data.

It is beyond the scope of this paper to evaluate which is the best function to be used for combining these scores in the computation of the overall score σ of an embedding. In general, σ is obtained as a function ϕ of the above properties: $\sigma = \phi(\gamma_1, \gamma_2, \gamma_3, \gamma_4, \gamma_5)$. The combination function ϕ can be derived, for instance, experimentally, or they might be specified by the user. As an example, ϕ can be a linear combination of its operands. Thus, additional flexibility can be reached by assigning *weigths* to each γ_i to denote the different importance of each property.

Thus, given an embedding $\tilde{e}$ for a query tree t_q in a document tree t_d, our ranking function ρ provides a *rich* relevance score, equipped with additional information on the grade of satisfaction of the above properties in $\tilde{e}$. This information helps the user to infer the overall *quality* of results. Further, the final score is enriched with some knowledge on key aspects like *completeness* and *cohesion* of data retrieved, usually neglected in scoring methods.

[5] Either parent-child or ancestor-descendant relationship.

4 Related Work

A recent initiative of the DELOS Working Group [1] emphasizes the growing importance of evaluating research methods against large collections of XML documents. To this end, approximate matching techniques are acknowledged to be powerful tools to be exploited. Thus, many proposals provide similarity query capabilities that, besides semantics, also examines the structure of documents to rank results [5, 7, 12, 13]. However, these approaches present relevance ranking functions that, in absence of knowledge of DTDs, either neglect some potential solutions or provide too coarse-grained scores that flatten the differences among actually different results. In XXL [13] partial match on query structure is not supported. Similarity basically depends on content similarity, through the evaluation of vague predicates. Flexibility on structural match is obtained through wildcards, but the number of skipped nodes is discarded in relevance computation, thus neglecting the grade of cohesion of data retrieved. XIRQL [7] introduces the concept of *index object* to specify document units as non-overlapping subtrees. Structural conditions only act as filters to determine the context where information has to be searched in. Partial match on query structure yet is not discussed. Few proposals consider partial match on query structure [5, 12]. In ApproXQL [12] similarity scores depend on the costs of basic transformations applied on the query tree. Basic costs are added in a total cost, thus resulting in a *absolute* scoring. Thus, it is not possible to realize the rate of query satisfied by an answer. A similar approach is proposed in [5]. Nodes and edges in a query tree are assigned pairs of weights that denote scores for exact and relaxed matching, respectively. Sum is used to combine scores of each single node/edge matching, yet producing an absolute scoring as in [12]. Despite the high flexibility provided by relaxations, also in this case information on cohesion of data is not taken into account for relevance. Further, to our knowledge, for a given document, all proposed methods return only the *best* (in some cases, approximate) matching. The set-oriented approach used in the $SATES$ function also captures the relevance given by multiple occurrences of the query pattern in the retrieved data.

Other related approaches deal with schema matching [10], and the semantic integration of XML data [8].

5 Conclusions and Future Work

We presented an *effective* relevance ranking measure to infer *quality* of results of similarity queries on XML data. Our measure widens the spectrum of relevant data, including solutions that also partially satisfy query requirements, and that approximate text organization inside documents. Then, our relevance ranking method provides a more fine-grained scoring, in that it takes into account the matching rate of query conditions, the cohesion of data retrieved, and the presence of multiple occurrences of query requirements inside data. Thus, besides correctness, a measure of completeness of query satisfaction, as well as knowledge about cohesion of results, are additional key elements to provide information on quality of data retrieved. Several issues urge to be investigated. In

order to augment query flexibility the user might be allowed to express preferences on either the semantics or the structure of a query. With regard to query processing, in order to cope with the possible hugeness of approximate results returned, we plan to exploit algorithms from works on top-k queries [6].

References

[1] Eval. Initiative for XML Doc. Retrieval. `http://qmir.dcs.qmw.ac.uk/XMLEval.html`. 33

[2] Extensible Markup Language. `http://www.w3.org/TR/2000/REC-xml-20001006`. 24

[3] XML Information Set. `http://www.w3.org/TR/xml-infoset`. 25

[4] XQuery 1.0: An XML Query Language. W3C Working Draft, 2001, `http://www.w3.org/TR/xquery`. 24

[5] S. Amer-Yahia, S. Cho, and D. Srivastava. Tree Pattern Relaxation. In *Proc. of the 8th Int. Conf. on Extending Database Technology (EDBT 2002)*, March 2002. 22, 23, 31, 33

[6] N. Bruno, L. Gravano, and A. Marian. Evaluating Top-k Queries over Web-Accessible Databases. In *Proc. of 18th Int. Conf. on Data Engineering (ICDE 2001)*, San Jose, CA, 2001. 34

[7] N. Fuhr and K. Großjohann. XIRQL: A Query Language for Information Retrieval in XML Documents. In *Proc. ACM SIGIR Conference*, 2001. 22, 31, 33

[8] E. Jeong and C. Hsu. Induction of Integrated View of XML Data with Heterogeneous DTDs. In *Proc. of 2001 ACM Int. Conf. on Information and Knowledge Management (CIKM 2001)*, Atlanta, USA, November 2001. 33

[9] P. Kilpeläinen. *Tree Matching Problems with Application to Structured Text Databases*. PhD thesis, Dept. of Computer Science, Univ. of Helsinki, SF, 1992. 24, 26

[10] J. Madhavan, P. A. Bernstein, and E. Rahm. Generic Schema Matching with Cupid. In *Proc. of the 27th VLDB Conf.*, pages 49–58, Rome, Italy, 2001. 33

[11] J. Robie, L. Lapp, and D. Schach. XML Query Language (XQL). In *Proc. of the Query Language Workshop (QL'98)*, Cambridge, Mass., 1998. 22

[12] T. Schlieder. Similarity Search in XML Data Using Cost-Based Query Transformations. In *Proc. of 4th Int. Work. on the Web and Databases (WebDB01)*, 2001. 23, 31, 33

[13] A. Theobald and G. Weikum. Adding Relevance to XML. In *Proc. 3rd Int. Workshop on the Web and Databases (WebDB 2000)*, pages 35–40, May 2000. 22, 31, 33

A New Path Expression Computing Approach for XML Data

Jianhua Lv[1], Guoren Wang[1], Jeffrey Xu Yu[2], Ge Yu[1],
Hongjun Lu[3], and Bing Sun[1]

[1] Northeastern University of China, Shenyang, China
{dbgroup,wanggr,yuge}@mail.neu.edu.cn
[2] The Chinese University of Hong Kong, Hong Kong, China
yu@se.cuhk.edu.hk
[3] The Hong Kong University of Science and Technology, Hong Kong, China
luhj@cs.ust.hk

Abstract. Most query languages in XML database systems use Regular Path Expressions (RPE) to query or extract data from databases and some query processing and optimization techniques have been proposed for RPEs. Conceptually XML documents are collections of path instances. Each path instance should conform to an XML element tag sequence, called path schema. A RPE query can be written as an automaton that can represent a language, while path schemas can be seen as sentences. In this paper, a novel RPE computing approach, automaton match (AM), is proposed. AM queries the RPEs by matching the automatons with path schemas. The experimental results show AM is quite efficient for computing RPE queries.

1 Introduction

With the widespread use of Web technologies and, in particular, the growing number of applications adopting XML as the standard language for representing and exchanging structured data on the WWW, managing XML documents through databases is becoming more and more important. There have been three main approaches to manage XML data, the relational, the object-oriented, and the special-purpose/native ways. Many relational storage methods for XML data have been devised, including the edge, attribute, region, inlining methods and SQL is used to query XML data. Similar to the relational way, the object-oriented way [9] stores XML into an OODB management system usually with classes and translates XML queries into OQL queries. In the third way, special structures, indices and particular query optimization techniques are applied.

A lot of query languages have been proposed to query XML data from databases [2]. A common feature of these languages is the use of regular path expressions (RPE). So how to rewrite and optimize RPE-based queries is becoming a research hotspot. A usual way is rewriting a RPE query with simple path expressions (SPE) based on schema information and statistics of XML data, then translate them into SQL queries, for example, in the relational way. In the

S. Bressan et al. (Eds.): EEXTT and DIWeb 2002, LNCS 2590, pp. 35–46, 2003.

Lore system [8], three basic query processing strategies are proposed: top-down, bottom-up, and hybrid. And XISS [5] also presented several algorithms based on the numbering scheme.

Altmel and Franklin proposed an algorithm to perform efficient filtering of XML documents in the selective information dissemination environments [1]. In their approach, automatons were used to represent user profiles, a SAX interfaced parser was used to parse XML documents and activate the automaton. However, in such systems, the roles of queries and data are reversed.

Path index is another special utility to evaluate path expression queries. We have developed a simple path index that can support the extent join algorithm [6], which can improve the performance of some very complex queries. Cooper and Sample proposed a fast index for semi-structured data that encoded paths as strings, and inserts those strings into a special index that is highly optimized for long and complex keys [3]. Nevertheless, the paths to be indexed should be user-defined since it is almost impossible to build index for every path.

In order to address the above problem, this paper presents a novel path expression evaluation approach named Automaton Match (AM) to efficiently compute path queries. The remainder of this paper is organized as follows. Section 2 introduces some basic concepts and ideas of XML and AM. Section 3 proposes two data structures, named path schema tree and path instance tree respectively, and shows how to evaluate RPE queries using them. Section 4 gives the performance evaluation on our experimental results. Finally, Section 5 concludes this paper.

2 Basic Concepts and Definitions

2.1 An Example

First, Fig. 1 gives an example XML document. Document Type Definition (DTD) can be used to constrain the information presented in an XML document. Fig. 2 is the DTD graph of the sample XML document shown in Fig. 1. Document Object Model (DOM) is an application programming interface (API) for XML and HTML documents. However, it can also act as a storage model. Fig. 3 gives the data tree of the document described in Fig. 1.

2.2 Path Schema and Path Instance

An XML tree can be decomposed into many path instances. The path instance in an XML tree is defined as the sequence of node identities from the root to a leaf or non-leaf node and the identities are linked by ".". The path schema of a path instance is defined as the sequence of tag names of nodes in the path instance linked by "/' except that the type #PCDATA is represented by a build-in function "text()" and attributes can only appear at the end of path expressions.

Some path schemas and path instances of Fig. 3 are given in Tab. 1. If all path schemas of an XML document can be gotten (actually it is easy to do), a RPE query can be handled by matching these path schemas with the RPE query.

```
<Document doc_id = "D1">
  <Title> T1 </Title>
  <Chapter Id = "C1">
    <Author> A1 </Author>
    <Head> CH1 </Head>
    <Section Id = "S1">
      <Head> SH1 </Head>
      <Paragraph> P1 </Paragraph>
      <Section Id = "S2">
        <Head> SH2 </Head>
        <Paragraph> P2 </Paragraph>
        <Paragraph> P3 </Paragraph>
        </Section>
    </Section>
    <Section Id = "S3">
      <Head> SH3 </Head>
      <Paragraph> P4 </Paragraph>
    </Section>
  </Chapter>
</Document>
```

Fig. 1. A sample XML document

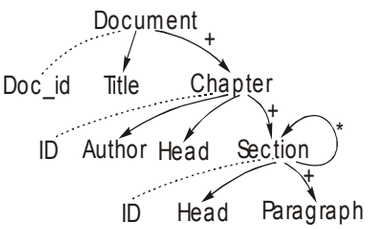


Fig. 2. The DTD of the sample XML

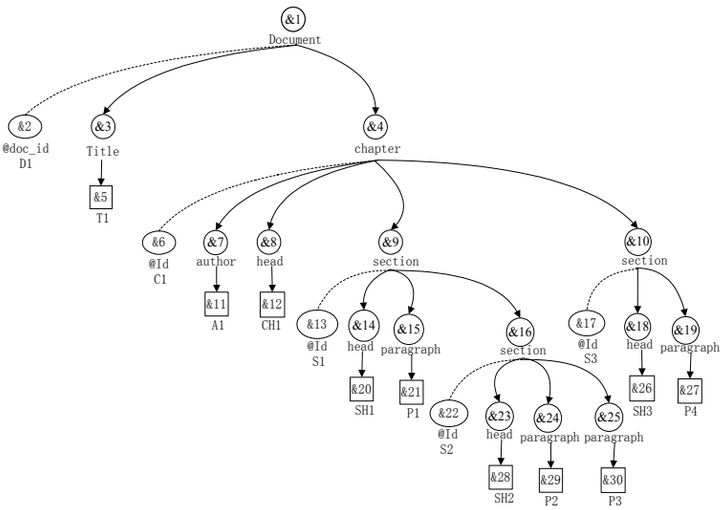


Fig. 3. The data tree of sample XML document

Table 1. Path schema vs. path instances

Path schema	Path instance
/Document	{&1}
/Document/Title/text()	{&1.&3.&5}
/Document/chapter/@Id	{&1.&4.&6}
/Document/chapter/section	{&1.&4.&9, &1.&4.&10}
/Document/chapter/section/section	{&1.&4.&9.&16}
/Document/chapter/section/section/@Id	{&1.&4.&9.&16.&22}
/Document/chapter/section/section/head	{&1.&4.&9.&16.&23}

2.3 Regular Path Expression Queries

A regular path expression query may contain four basic path operators, namely parent-child, ancestor-descendant, closure, and predicate filtering operators. As the ancestor-descendant operator can be represented by the other three operators, so RPE query can be defined as Fig. 4.

The following gives two RPE query examples following the rules in Fig. 4 based on the XML tree shown in Fig. 3.

Query1: Find out head of all chapters.
RPE Query: /Document/chapter/head
Query2: Find out all paragraph of section S2 in chapter C1.
RPE Query: /Document/chapter[@Id=C1]/section[@Id=S2]/paragraph

2.4 Query Finite State Automaton

A RPE query can be represented as a Finite State Automaton (*FSA*) [1]. A RPE *FSA* must comply with the syntax definition rules given in Fig. 4 and can be a quintuplet form: $RPEAutomaton ::= < K, \Sigma, \delta, q_0, F >$, where K is a set of all states including the start state, the end states set and the intermediate states, in which each state can have incoming or outgoing edges named element names or attribute names. Σ is an alphabet that is in fact a set of actions that can be

```
RPEquery ::= '/' RPEquery        // the root element
    | RPEquery '/' RPEquery      // parent-child operator
    | RPEquery '*'       // closure operator
    | RPEquery '+'       // closure operator
    | (RPEquery)        // to change PRI of operators
    | step
step ::= name        // Element name
    | '@'name        // Attribute name
    | name '['predicate']'       // Predicate on element
```

Fig. 4. BNF syntax of RPE

Input: RPE FSA fsa represented user RPE query Path schemas and path instances of a XML document to be queried
Output: a result set that contains element instances fulfilling the given RPE query
Algorithm body:
(1) Set the result set to be an empty set
(2) For all path schemas ps in the target XML document
(3) Do
(4) If ps is accepted by fsa
(5) Then insert the end element instances of path instance pi corresponding to ps into result set
(6) endif
(7) Done

Fig. 5. The Automaton Match algorithm

applied on the set of states K. δ specifies the state transfer function. And q_0 is the start state, which is a particular state that has only one outgoing edge, the document element of a DTD. Finally, F is the end state set. The input of an action is one state and the output may be either another state or the same state. An action can be either element name or attribute name. The parent-children operator in a RPE query is transformed into a single step from a state in RPE *FSA* to another, and closure operators (* or +) are then represented by the cycles in *FSA*.

2.5 Automaton Match

Since all RPE queries can be converted into RPE *FSAs*, all XML data can be seen as collection of path instances and each path instance corresponds to a path schema that could be read by RPE *FSA*, so we got the *Automaton Match* algorithm to evaluate RPE queries. The following (Fig. 5) is the general Automaton Match algorithm written by pseudocodes.

Fig. 5 shows how Automaton Match evaluates user RPE queries. However, its performance may not be very efficient because the path schemas one XML document contains could be a very large amount. So the following section will introduce some index structures in order to improve the performance.

3 Structures and Algorithms

In *AM*, path schemas are used to match the RPE *FSAs* and the matching results are the element instances of the end parts of path instances. So the mapping from path schemas to path instances is one of the main problems of *AM*. A straightforward implementation is indexing the relationship between path schemas and path instances. Although the it is correct from the viewpoint of functionality, it is the worst way from the viewpoint of performance since (1) for

n path schemas the *FSA* has to be matched n times; (2) a great deal of common prefix are stored and repeatedly but unnecessarily handled and therefore extra overhead occurs. In the following subsections, index structures are introduced to improve the performance.

3.1 Path Schema Tree

The path schema tree is used to index the path schemas of an XML document to be queried. All path schemas in an XML document are stored together with one *PST*, which is used as sentences to match the *FSA* of the RPE query. It is derived from the path instances of an XML document and not from a DTD. A *PST* actually likes a trie tree if we treat path schemas as strings and element names (attribute names) as characters in a string. The nodes of *PST* are element names and edges specify the nesting relationships between elements. Then each node in the *PST* tree is dealt with only once and extra overhead occurring in the straightforward way is avoided. Fig. 6 shows the *PST* of the sample document in Fig. 1. The numbers inside the nodes can be regarded as path schema identities. The solid lines with arrows between nodes represent the parent-children relationships; dotted lines represent the element-attribute relationships, while the rectangle nodes mean that they are text nodes. Each node in the path schema tree corresponds to a path schema.

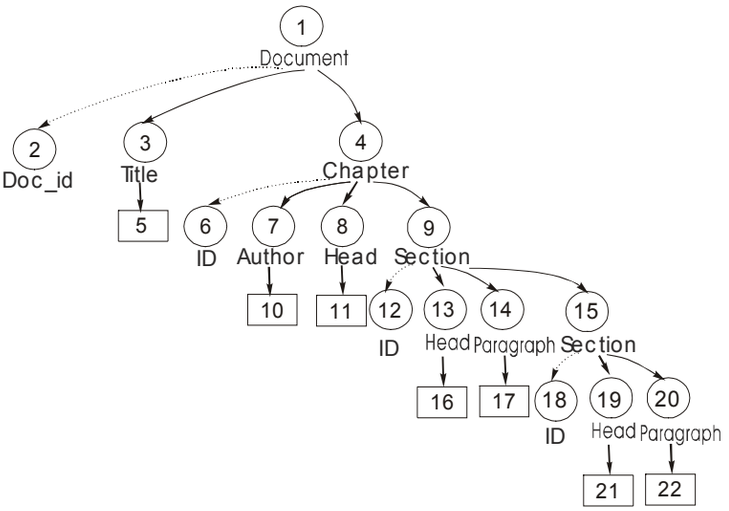


Fig. 6. Path schema tree of sample document

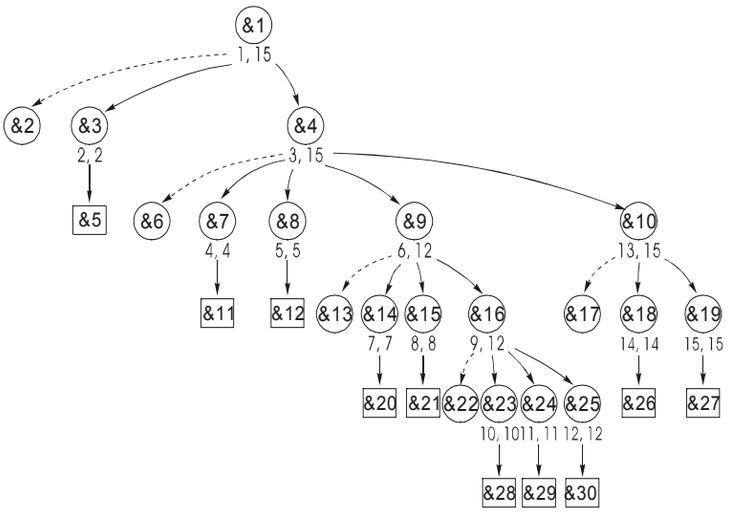


Fig. 7. Path instance tree of sample document

3.2 Path Instance Tree

Similarly, all path instances in the XML document are stored together with a *path instance tree* (*PIT*). *PST* is used to match the RPE *FSA*, and *PIT* is used to retrieve results. Fig. 7 shows the *PIT* of the sample document. The numbers inside the nodes are element/attribute identities and the meaning of lines and nodes are the same as Fig. 6.

Conceptually the combination of *PST* and *PIT* is very similar to the Dataguides for they two are all derived from the XML data. However there are some differences between them. First, Dataguides are designed to act as the schema of semi-structured data and assist querying, while these two indexes are designedly query supporting structures. Second, Dataguides do not maintain the relationship between the element instances that are organized into different target sets. In our approach, the path schemas and path instances are stored separately (*PST* and *PIT*) and the relationships are all preserved for better query performance.

3.3 Automaton Match Algorithm

With *PST* and *PIT*, the algorithm shown in Fig. 5 can be written to a more detailed one. Actually, in *AM* approach, the RPE *FSA* is matched not by a sentence but by a tree (path schema tree). So the matching algorithm of *AM* is much different from the common *FSA* matching algorithms.

The *AM* algorithm shown in Fig. 8 uses a list (or said stack) to store states and words (element names) ready for processing, therefore, which action will happen first could not be controlled forehead on purpose. In the *AM* algorithm, every nodes of the *PST* should be used to match the states of the query automaton, and the states that the nodes matched are marked in the *status* fields of nodes. *AM* processes the *PST* from the root node. Tab. 2 simulates the executing steps of the query 2.

```
Input:
    RPE FSA fsa
    Path Schema Tree pst
    Path Instance Tree pit
    Mapping Table mt
Output:
    result set rs
Algorithm body:
pst.root.status ← fsa.startstatus;        // set the status of pst's root
waitlist ← {pst.root};        // the word to be read is the root of pst
while waitlist is not empty
    currentnode ← shift (waitlist);     // pop a word from waitlist as if it is a stack
    if has predicate on currentnode then
        mark the exclusive sub-instancetree
                                        //filtering the element instances with predicates
    endif
    if currentnode.status is fsa.finalstatus then        // the end state of fsa
        for each instance in currentnode.instanceset do
                                                  // using mt to get instances from pit
            if instance has not be marked then        // result checking
                rs ← rs {instance};              // this element instance is a result
            endif
        done
    endif
    for each child of currentnode cc do
        if fsa has a state transition from state labelled currentnode.status to a status
          ts activated by cc then
            cc.status ← ts; // set the children's status
            waitlist ← waitlist ∪ {cc}; // push a word into waitlist
        endif
    done
endwhile
```

Fig. 8. Detailed AM algorithm

Table 2. Example of Automaton Matching

	Word read	Current fsa state	Current PST node	Waitlist	Actions
1		0		{1}	
2	document	1	1	{2, 3, 4}	
3	Chapter [@ID="C1"]	2	4	{6, 7, 8, 9}	Mark subtree (3, 15)
4	Section [@ID="S2"]	3	9	{12, 13, 14, 15}	Mark subtree (6, 12)
5	paragraph	4	14	{17}	
6	text()	5		{ }	Get Results {21}

4 Performance Evaluation

4.1 An Overview

In this section, we will discuss the performance evaluation of the *AM* with respect to two benchmarks. The experiments were made on a single 933MHz CPU PC with 384MB main memory. We employed a native XML management system called XBase [6] as the underlying data storage, which stores XML document into an object database through an ODMG-binding DOM interface. The testing programs were coded with MS VC++ 6.0 and Inada 2.0(A persistent object-oriented programming language). The datasets we used are listed as follows:

XMark: The first data set is from XML benchmark project [10]. The scale factor we selected is 1.0, and the corresponding XML document size is 100 Mega Bytes.

Shakes: The second data set is the Bosak Shakespeare collection.

In order to full explore the performance of *AM* algorithm proposed in this paper, we implemented three different query evaluation methods: *DOM*, *EJ* and *AM*. The *DOM* approach evaluates path expressions by traversing the XML data tree (DOM tree) from top to end with no index is used, which is similar to the top down approach in Lore system. The extent join approach (*EJ*) decomposes the RPE queries into several atomic path fragments, temporary results can be computed out for each fragments, then a parent-children relationship based multi-join is performed to get the final results. To be as fare with other approaches as possible, in our experiments only the structure indices are used to support *EJ*.

4.2 Experiments and Performance

The tests on XMark queries are performed. The results show that for most queries the *DOM* approach is the worst one since it should traverse the data tree stored in database and therefore need more disk I/Os. It seems that *AM*

outperforms the other two approaches in the cases of long, complex path expressions. However, since the queries of XMark are all designed to test XQuery query processing performances, there are too many complex, high cost operations other than the RPE operators, such as the join on values, reconstruction, construction of complex results, and etc. In order to clearly evaluate the performance of these three approaches on RPE, we extracted 18 RPEs from the XMark queries to do further experiments. For the limitation of space, the paths are omitted in this paper.

Fig. 9 illustrates the performance of different path expression computing approaches over different paths. From the experiments results, we can see that *DOM* is almost the worst path expression evaluation approach (slowest on 16 RPEs). This is exactly the reason we want to design new algorithms. The average evaluating speed of *EJ* and *AM* is about 2 to 20 times, sometimes can be hundreds of times, faster than that of *DOM*. The performance comparison between *DOM* and *EJ* is not in this paper's emphases, so we concentrate on the *AM* approach in the following analysis.

According to the experiments results, the performance of these algorithms can be classified into several categories: 1) For P8.2, *EJ* outperforms the other two and *AM* is the worst. P8.2 is a short path and the result number is much small (97), so only 2 joins was needed by *EJ* and *DOM* traversed only a relatively very small part of data tree. P8.1 is also a short path, while is this case, *AM* performs better than *DOM* for its result number is larger (255) and this path expression demands *DOM* to traverse a bigger part of data tree; 2) For P1, P5, P12.1, P20.1, P20.2 and P20.3, the performance of *AM* is not very distinctive, about 30% to 400% faster than *DOM* and 20% to 40% faster than *EJ*. The reason is that these RPEs all have predicates in and all these three approaches need to access database to perform predicate filtering, so the performance of *AM* is drown; 3) The RPEs with moderate length (3 to 5 steps) and have no predicates in can be divided into two groups: one group contains P2, P6, P9.3 and P13, the performance of *AM* on them is slightly slower than *EJ* (20% to 30%) and faster than *DOM* (2 to 10 times); the other group contains P10, P11.1, P11.2 and P15, where *AM* outperforms the other two methods. It is confusing, and after analyzing the steps of these paths that automaton walked by, we found it might be the peculiar tag names acting as steps that made the difference.

Another conclusion we can draw is: the length of path expression does affect the *AM* performance. Let us look at P8.1 and P11.1 respectively, P11.1 has just one more step "profile" than P8.1. As the result, *AM* performs worse than *EJ* on P8.1 and better on P11.1; 4) For the rest RPEs (P7 and P18), *AM* performs greatly better than the others, averagely 40 to 120 times faster than *DOM* and 3 to 6 times faster than *EJ*. Especially for P15, *AM* is 226 times faster than *EJ*. However, the reasons are different: a) P7 is a complex path expression which contains "|" operator and "//" operator that could be rewritten to several branches, while P15 is a very long path expression.

Another data set we used is Shakespeare's drama. We used the same queries as XRel. Fig. 10 shows the performance on these queries. Q2 is a long path

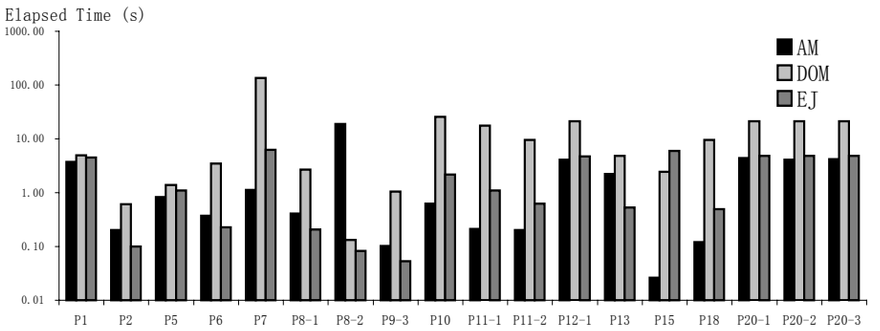


Fig. 9. Performance of RPEs of XMark (100M)

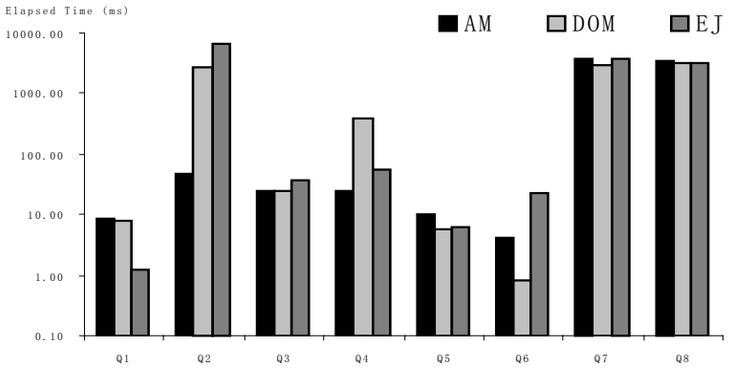


Fig. 10. Performance of Shakes

while Q3, Q4 and Q6 are complex paths with path branches or "—" operators, so *AM* outperforms *DOM* and *EJ* on these queries. There are little difference between *DOM*, *EJ* and *AM* on Q7 and Q8. This is because these two queries all have predicates, while all the three approaches need access database to filtering elements.

5 Conclusions

In this paper, we proposed a new RPE evaluation approach named Automaton Match (*AM*) for XML data. In *AM*, XML data are abstracted into path instances and indexed by *PIT*. Each path instance corresponds to a path schema that is an ordered list of element names, and path schemas in an XML document are

organized with a *PST*. User RPE queries are converted into *FSAs*. The query evaluating procedure is matching the RPE *FSA* with the *PST*. After the end states of *FSA* are reached, the corresponding path instances are checked and retrieved as query results. Our experiments result show that *AM* performs well.

However, the performance of the predicate filtering operators is not so good. As a future work, integrating signature technique into *AM* may be a challenging work. Another interesting issue that need more research work is how to support the ancestor-decedent connectors ("//") directly with *AM*.

Acknowledgements. This work was supported by the National Research Foundation (60273079) and the Teaching and Research Award Program for Outstanding Young Teachers in Higher Education Institutions of the Ministry of Education of China.

References

[1] M. Altmel and M. Franklin. Efficient filtering of XML documents for selective dissemination of information. Proc. of the 26th VLDB Conf., Cairo, Egypt, 2000, 53-63. 36, 38

[2] A. Bonifati and S. Ceri. Comparative analysis of five XML query languages. SIGMOD Record, 2000, 29(1): 68-79. 35

[3] B. F. Cooper, N. Sample, M. J. Franklin, G. R. Hjaltason and M. Shadmon. A Fast Index for Semistructured Data. Proc. of the 27th VLDB Conf., Roma, Italy, 2001, 341-350. 36

[4] R.Goldman and J. Widom. DataGuides: Enabling Query Formulation and Optimization in Semistructured Databases. Proc. of the 23rd VLDB Conf., Athens, Greece, 1997, 436-445.

[5] Q. Li and B. Moon. Indexing and querying XML Data for regular path expressions. Proc. of the 27th VLDB Conf., Roma, Italy, 2001, 361-370. 36

[6] H. Lu, G. Wang, G. Yu, Y. Bao, J. Lv and Y. Yu. Xbase: Making your gigabyte disk queriable. Proc. of the 2002 ACM SIGMOD Conf. 2002. USA. 36, 43

[7] J. McHugh, S. Abiteboul, R. Goldman, D. Quass, and J. Widom. Lore: A Database Management System for Semistructured Data. SIGMOD Record, 26(3):54-66.

[8] J. McHugh and J. Widom. Query optimization for XML. Proc. of the 25th VLDB Conf., Edinburgh, Scotland, 1999, 315-326. 36

[9] A. Renner. XML Data and Object Databases: A Perfect Couple? Proc. of the 17th ICDE Conf., Heidelberg, Germany, 2001, 143-148. 35

[10] A. Schmidt, F. Waas, M. Kersten, M. J. Carey, I. Manolescu and R. Busse. XMark: A Benchmark for XML Data Management. Proc. of the 28th VLDB Conf., Hong Kong, China, 2002. 43

[11] M. Yoshikawa, T. Amagasa, T. Shimura, and S. Uemura. XRel: A path-based approach to storage and retrieval of XML documents using relational databases. ACM Transactions on Internet Technology, 2001, 1(1).

Integrated XML Document Management

Hui-I Hsiao, Joshua Hui, Ning Li, and Parag Tijare

IBM Almaden Research Center
650 Harry Road, San Jose, California, USA
{hhsiao,jhui,ningli,tijare}@us.ibm.com

Abstract. XML has become a widely accepted data format for exchange and representation of semi-structured data. Efficiently managing XML as well as traditional business documents and content in an integrated fashion is a necessity for companies to be successful in the e-generation. Many projects [14], [15], [18] have been developed for managing XML documents. Nevertheless, these projects focus mainly on storing, searching, and retrieving XML documents. No solution has been proposed to manage and query both the XML documents and other traditional text and media data in an integrated fashion. In this paper, we present our research effort in building an XML document management system that manages both XML and non-XML content in a uniform way. Our focus is not only on managing XML documents, but also on providing a uniform view of and integrated access to XML and non-XML documents.

1 Introduction

Business content is more than structured business records. It is typically considered to be the information that supports business operations, including not only structured business data, but also all the supporting semi-structured and unstructured data such as product specifications, product images, service and support information, etc. Since the size of the semi-structured and unstructured data is often very large and data have to be accessed and delivered in an efficient manner, those data are not stored inside database systems normally but in file systems or specialized servers such as video-on-demand streaming servers and hierarchical storage systems [1], [3], [7]. To fully capture the semantics behind semi-structured and unstructured data and to provide efficient search capability, various technologies (e.g. QBIC [6], text indexing, classification, clustering, and search technologies) have been developed to extract and index the metadata information from the data.

A content management system is designed to discover, build, integrate, and manage the data with the corresponding metadata. In such a system, data is normally stored in a data server while metadata associated with the stored data is extracted, indexed, and stored in a metadata server (a DBMS normally). While the metadata server and the data server (also referred to as an Object server) functions can be built in a single

S. Bressan et al. (Eds.): EEXTT and DIWeb 2002, LNCS 2590, pp. 47-67, 2003.

computer system, it is advantageous to have the metadata server separated from the data server such that a metadata server can be associated with multiple data servers and each of the data servers can be placed close to the set of users that most frequently access a server. This allows centralized searching with distributed and efficient data delivery to users.

Even though the data and the corresponding metadata would reside in multiple systems across different locations, they are viewed as one entity. Both referential integrity and data consistency among the data are ensured and update to both resource data and metadata are performed in a transactional manner. Content management systems have increasingly become a necessity rather than luxury for companies to be competitive and successful today. This is especially true as more and more enterprises are moving towards web-based commerce and business models. Commercial content management systems that provide integrated business content management function include IBM Content Manager [10], Oracle 9i [16], Microsoft Content Management Server [13], Documentum 4i [5], and FileNet Panagon [8].

Besides the traditional semi-structured and unstructured data such as text and image files, there is growing importance of Extensible Markup Language (XML) documents. Because of its self-describing nature, XML has become a widely accepted data format for exchange and representation of semi-structured data. XML is also used as a basis for capturing the meta information of unstructured data, such as video and audio. One example is the MPEG-7 standard [12] for describing the various types of multimedia information. The description is associated with the content, to provide a better searching capability on the data itself. Thus, managing XML documents efficiently has recently become one of the major focus areas in the academic research and industrial communities [14], [15], [18]. Nevertheless, to the best of our knowledge, these projects focus mainly on storing, searching, and retrieving XML documents, and in some cases, managing XML with the business records stored inside databases. No solution has been proposed on how to manage and query both the XML data/documents and other traditional text and media data in an integrated fashion. Since XML documents will be only one part of enterprise business content, providing a uniform view of and integrated access to both XML and non-XML content is as important as efficient management of XML documents.

In this paper, we describe our research effort in building an XML document management system that provides a uniform view to both XML and non-XML documents. In our system, referential integrity and transaction consistency are provided between XML data and its associated text and media data. Our system supports XQuery [21] as the query language to allow aggregate and join queries to be issued across data of all types. Moreover, features that are usually provided by a content management system, such as linking and reference are also explored to bridge the gap between XML data and other semi-structured and unstructured data.

The rest of the paper is organized as follows. Section 2 describes related work in the XML document management area. Section 3 gives an overview of a traditional content management system. Section 4 describes our prototype that provides XML and non-XML document management in an integrated fashion. Section 5 shows some performance measurements. Lastly, we summarize our experience and conclude our work in Section 6.

2 Related Work

Many approaches have been proposed to store and query XML documents using a Relational Database Management System (RDBMS). Among them, [18] exploited the schema information in DTDs and developed three inlining techniques to improve query performance. In [2], a relational storage mapping is generated, which includes decisions of inlining/outlining elements and attributes, based on XML Schema, XML data statistics and XML query workload. In [4], a relational storage model with a semistructured overflow graph is generated automatically from a collection of schema-less semistructured instances, using data mining techniques. In [9], schema-less XML documents are viewed as graphs with nodes as elements and edges as parent-child relationships. Different storage schemes of the edges are presented with their performance evaluated.

Several traditional content management systems, such as Documentum [5] and iManage [11], have claimed to provide the functionality for supporting the storage of XML documents. However, to the best of our knowledge, none of them supports an XML query language yet.

Our work differs from previous work in that, firstly, we store and query (using an XML query language) XML documents conforming to XML Schemas using a Content Management System, where a hierarchical data model is commonly adopted. Secondly and more importantly, we provide an integrated solution to manage both XML and non-XML data, which includes storing, indexing, querying, retrieving and linking XML and non-XML data.

3 Overview of Content Management Systems

A Content Management System (CMS) is a repository for virtually any type of digital content, including HTML and XML web content, document images, electronic office documents, and rich media such as digital audio and video. A single CMS typically supports multiple content (data) stores distributed across the enterprise, or across the internet. This allows content to be stored close to its point of use but under central management control, reducing bandwidth requirements and increasing disaster protection. The functions provided by a CMS typically include check-in/check-out, version control, object-level access control, and advanced searching, workflow, automatic routing and tracking of content through a business process according to predefined rules.

3.1 CMS Architecture

CMS architecture is commonly based on a triangular client/server model comprised of a metadata server, one or more object (or data) servers, and one or more clients. IBM Content Manager [10], FileNet Panagon [8], and Documentum 4i [5] are examples of content management systems based on this architecture. In the IBM Content Manager, for example, a component called Library Server (the metadata server) manages the

catalog and metadata information, locates stored objects using a variety of search technologies, and provides secure access to the objects held in the collection. It also coordinates distributed transactions spanning the Library and Object Servers to provide data integrity. The digital content is stored in the Object Servers, which can be distributed across an enterprise network to provide convenient user access. This is especially important for delivery of large text files and multimedia objects, such as audio and video. Figure 1 shows an example of typical content management system architecture.

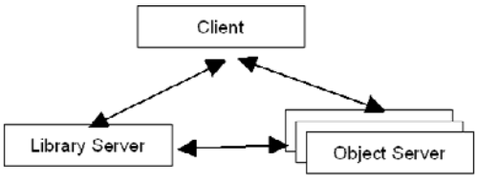


Fig. 1. Content Management System Architecture

In the following section, we will use the IBM Content Manager as an example to describe the data model and storage architecture of a typical content management system.

3.2 Metadata Server Data Model

The Library Server component of the IBM Content Manager supports a rich data model for storing catalog and metadata information. An Item is the basic unit of resource managed by the Library Server. An Item is a typed object whose type is defined by an Item Type. Logically, an Item Type comprises Component Types arranged in a hierarchy. This hierarchy forms a tree structure and has a unique root Component Type. An Item is an instance of an Item Type. It comprises one instance of the root Component Type and zero or more instances of descendent Component Types. It is possible to define the minimum and maximum number of occurrences of the instances of a non-root Component Type at the time of its definition. Within the Item, these component instances have ancestor-descendent relationships as dictated by the Item Type definition. In implementation, an Item Type comprises multiple relational database tables, each representing a Component Type of the Item Type. An Item comprises one row from the database table representing the root Component Type and zero or more rows from each of the database tables representing descendent Component Types. When an application creates an Item, the Library Server assigns it several system-defined attributes, including an Item identifier. The Item identifier is used to locate the Item within the Library Server. The Library Server also allows applications to create application-specific attributes. Items can be associated with one another via links. A link describes a relationship between two Items or a relationship of an Item to itself. Link relationships are represented in a separate relational table. Each row in this table identifies the source and the target Item of a link. In addition, the data model supports reference attributes that can reference other Items. Reference attributes can be defined to be a part of any Component Type within an Item Type. Reference attrib-

utes are implemented as a set of columns in Component tables they belong to. These columns together identify the row and the table corresponding to the target Item.

Figure 2(a) illustrates the Library Server data model using an example Item Type to store information about technical journals. This Item Type has four Component Types: Journal, Editor, Article and Author. Journal is the root Component Type. Each Component Type is represented by a relational table. Journals can have multiple editors and articles; articles can have multiple authors. The table hierarchy depicts this structure. The ComponentId and ParentComponentId columns capture parent-child relationships between components. The data in Figure 2(a) shows, for example, that IBM Journal of Research and Development, Volume 45, Number 5 has Advanced Semiconductor Lithography as its topic, with George J. Hefferon as its editor and contains two articles on pages 605 and 651, etc. Figure 2(b) shows Item Type data for resource items, which store information about the location of objects on Object Servers. Figure 2(c) shows Link relationships. The data in Figure 2(c) shows, for example, that the content of the IBM Journal of Research and Development is stored in pdf format in the file /journals/resanddev.pdf on an Object Server with id 1.

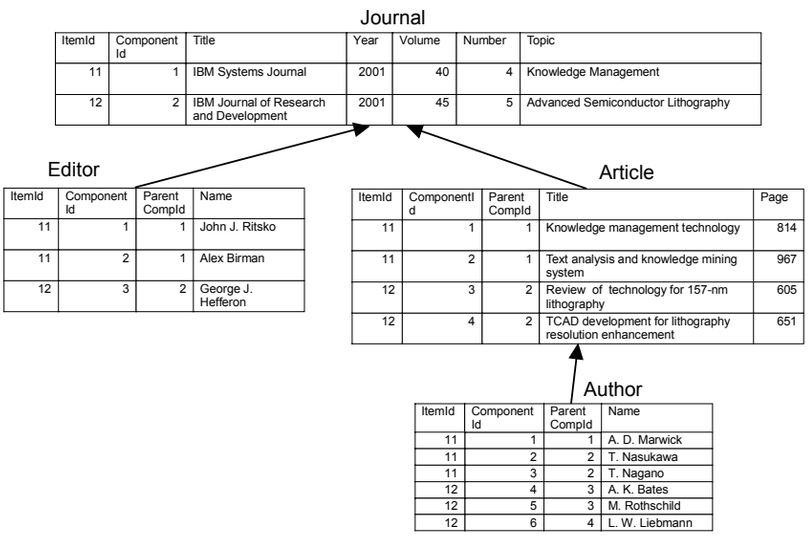


Fig. 2(a). Example of Library Server data model

Location

ItemId	ComponentId	File Name	Object ServerId
21	1	/journals/resanddev.pdf	1
22	2	/journals/systems.pdf	1

Fig. 2(b). Item Type data for resource Items

Link

Link ItemId	Source ItemId	Target ItemId	LinkType
31	11	22	pdf-content
32	12	21	pdf-content

Fig. 2(c). Link relationships

3.3 Integrated CMS

Content management systems have been used to manage enterprise documents for more than a decade. Besides providing a rich data model for modeling/capturing complex business "objects" (items), many content management systems, such as the ones from Documentum and Microsoft, also provide document authoring and publishing functions for web site content creation and deployment. As XML is increasingly becoming the de facto standard for content publishing and exchange, the number of XML documents generated and published has increased dramatically over the past few years. The percentage of businesses using or planning to use XML has increased from 20% in year 2000 to 78% in 2001, according to a survey sponsored by vendor XMLSolutions. Zona Research predicts that the percentage of e-commerce transactions using XML was 0.5% in the year 2000 and is expected to jump to 40% by the end of 2003. In the foreseeable future, XML is expected to be the primary format, if not the only format, of all new business documents in most enterprises and institutions. Consequently, it is important for content management systems to also manage XML documents efficiently. However, managing XML documents as plain text documents is not sufficient, a CMS needs to be able to exploit the rich semantics of the XML documents to provide more efficient storage, indexing, and search functions. Furthermore, XML documents are parts of overall business content. For companies and institutions that already have Terabytes or even Petabytes of data/content managed by content management systems, it is desirable to have both XML and non-XML documents managed uniformly and in an integrated fashion, preferably in the same CMS, such that users can easily search and retrieve information in both types of documents.

In the following section, we will describe our research effort in building an efficient and **integrated** XML document management system by exploiting and extending current CMS technologies.

4 Integrated XML Document Management

Like traditional document management, the set of functions needed for XML document management include store, index, search/query, and retrieval of the documents. Our XML management system currently supports XML documents with schema de-

fined in the XML Schema language [23]. Schema-less document management is a subject for future study. To store an XML document, the corresponding XML schema will need to be loaded first. In the following, we will first describe XML schema loading and mapping to the Library Server data model. Section 4.2 describes the storage model and management of XML documents such that efficient search as well as partial fragment and full document retrieval are provided. Section 4.3 discusses XML query support that provides a uniform and integrated search function across XML and non-XML documents. Finally, Section 4.4 discusses link and reference support for managing relationship among XML documents, which is a necessary function for integrated business content management.

4.1 Schema Mapping

Our system supports storing and querying XML documents conforming to XML schemas. The XML Schema language is a new standard for defining the schema of XML documents and it can be used to generate efficient indexing and storage strategies.

4.1.1 Simplifying XML Schema

Because we have assumed an unordered data model, as in many other works, e.g. [2], [4], [9], [18], an input XML schema is simplified before it is mapped to the Library Server data model when the schema is to be loaded into our system. More specifically, transformations are applied to simplify the nested "element", "group", "all", "choice" and "sequence" nodes in element type definitions.

First, we define a representation transformation *f* which simplifies the representation for the kinds of nodes mentioned above, as follows:

f('<element name=***e*** type=***d*** minOccurs=***min*** maxOccurs=***max*** />') = ***e***[***min***:***max***]
f('<all minOccurs=***min*** maxOccurs=***max***>' ***x*** '</all>') = *f*(***x***)[***min***:***max***]
f('<choice minOccurs=***min*** maxOccurs=***max***>' ***x*** '</choice>') = *f*(***x***)[***0***:***max***]
f('<sequence minOccurs=***min*** maxOccurs=***max***>' ***x*** '</sequence>') = *f*(***x***)[***min***:***max***]
f('<group>' ***x*** '</group>') = *f*(***x***)[1:1]
f(***x y***) = *f*(***x***), *f*(***y***)

where ***x*** is an XML schema fragment, ***e*** is the *name* of an element, ***t*** is the *type* of the element, ***min*** and ***max*** are the values of *minOccurs* and *maxOccurs*, respectively, which can take default values. And *maxOccurs* could be of value "unbounded". An example of the transformation is as follows:

```
f('<sequence maxOccurs="3">
        <element name="A" type="string" minOccurs="0"/>
        <element name="B" type="string"/>
        <choice maxOccurs="unbounded">
                <element name="C" type="string"/>
                <element name="B" type="string"/>
        </choice>
 </sequence>')
```

= *f*('<element name="A" type="string" minOccurs="0"/>'
'<element name="B" type="string"/>'
'<choice maxOccurs="unbounded">
<element name="C" type="string"/>
<element name="B" type="string"/>
</choice>')[1:3]
= (*f*('<element name="A" type="string" minOccurs="0"/>'),
f('<element name="B" type="string"/>'),
f('<choice maxOccurs="unbounded">
<element name="C" type="string"/>
<element name="B" type="string"/>
</choice>'))[1:3]
= (A[0:1], B[1:1],
(*f*('<element name="A" type="string"/>'),
f('<element name="B" type="string"/>'))[0:unbounded])[1:3]
= (A[0:1], B[1:1], (C[1:1], B[1:1])[0:unbounded])[1:3]

Then, we apply three types of transformations to simplify the element instances, as shown in Table 1.

Table 1. Transformations to simplify element instances

Element Instance	Simplification
e[min1,max1] [min2,max2]	e[min1× min2,max1× max2]
(e1[min1,max1], e2[min2,max2]) [min3,max3]	e1[min1× min3,max1× max3], e2[min2× min3,max2× max3]
..,e[min1,max1], ...e[min2,max2],	...,e[min1+min2,max1+max2],

These three transformations are similar to the simplification, flattening and grouping transformations presented in [18]. The essential information, i.e. the parent-child relationship, is preserved after the transformations. Here is the result of applying these transformations to the previous example:

(A[0:1], B[1:1], (C[1:1], B[1:1])[0:unbounded])[1:3]
= (A[0:1], B[1:1], C[0:unbounded], B[0:unbounded])[1:3]
= (A[0:1], B[1:unbounded], C[0:unbounded])[1:3]
= A[0:3], B[1:unbounded], C[0:unbounded]

4.1.2 XML Schema Graph

Given a simplified XML schema, we generate an XML schema graph. An XML schema graph represents the structure of the schema. Its nodes are elements, attributes, complex types and simple types in the schema. Each element or complex type node appears only once. The graph's element to element edges and complex type to element edges are annotated by the minOccurs number and the maxOccurs number. Our system currently does not support schemas with mutually recursive element (or complex type) nodes or element (or complex type) node with an in-degree greater than one.

Technologies similar to those developed in [2] and [18] can be applied to remove these limitations.

We further simplify the schema graph by applying the following operations:

1) remove all complex type nodes:
   for each complex type node **Nc**
      for **Nc**'s incoming edge **E1** from element node **Ne1**
      and each outgoing edge **E2** of **Nc** to element node **Ne2** with annotation [**min**,**max**]
        create a new edge **E** from **Ne1** to **Ne2** and annotate **E** with [**min**,**max**]
      then remove **Nc** and its edges
2) replace simple type nodes with their corresponding primitive type nodes:
    for each simple type node **Ns**
      find its base primitive data type **p**
      then replace **N**s with a primitive type node **Np**

Figure 3(a) shows an XML schema file Journal.xsd and Figure 3(b) is the schema graph generated from Journal.xsd. For simplicity of the graph, no simple type nodes are shown. Figure 3(c) shows the schema graph after complex type nodes removal and simple type nodes replacement. Again for simplicity, no primitive type nodes are shown on the graph.

```
<?xml version="1.0" encoding="utf-8" ?>
<xsd:schema
        xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <xsd:element name="Journal" type="journalType" />
    <xsd:complexType name="journalType">
        <xsd:sequence>
            <xsd:element name="Editor" type="editorType"
                                maxOccurs="10" />
            <xsd:element name="Article" type="articleType"
                                maxOccurs="unbounded" />
        </xsd:sequence>
        <xsd:attribute name="Title" type="xsd:string" />
        <xsd:attribute name="Year" type="xsd:integer" />
        <xsd:attribute name="Volume" type="xsd:integer" />
        <xsd:attribute name="Number" type="xsd:integer" />
        <xsd:attribute name="Topic" type="xsd:string" />
    </xsd:complexType>
    <xsd:complexType name="editorType">
        <xsd:attribute name="Name" type="xsd:string" />
    </xsd:complexType>
    <xsd:complexType name="articleType">
        <xsd:sequence>
            <xsd:element name="Author" type="authorType"
                                maxOccurs="20" />
        </xsd:sequence>
        <xsd:attribute name="Title" type="xsd:string" />
        <xsd:attribute name="Page" type="xsd:integer" />
    </xsd:complexType>
    <xsd:complexType name="authorType">
        <xsd:attribute name="Name" type="xsd:string" />
    </xsd:complexType>
</xsd:schema>
```

Fig. 3(a). Example XML Schema

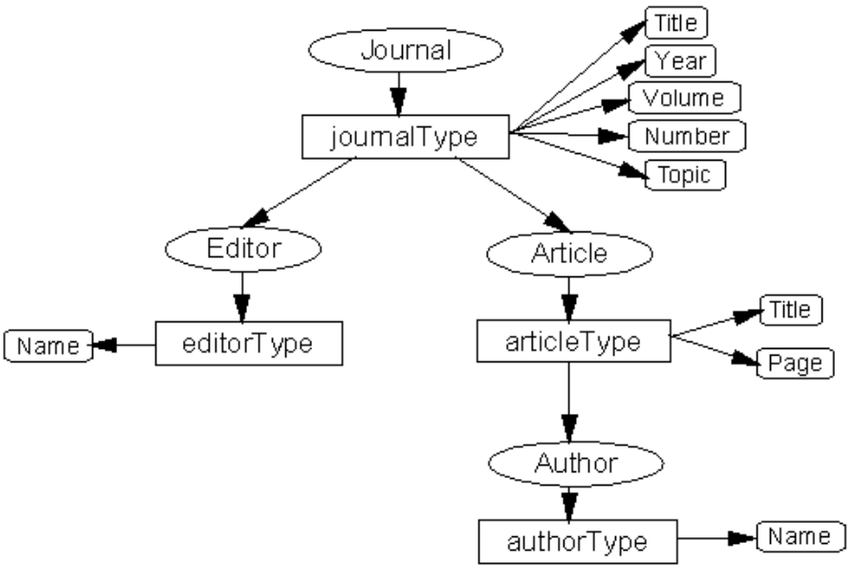


Fig. 3(b). Schema Graph

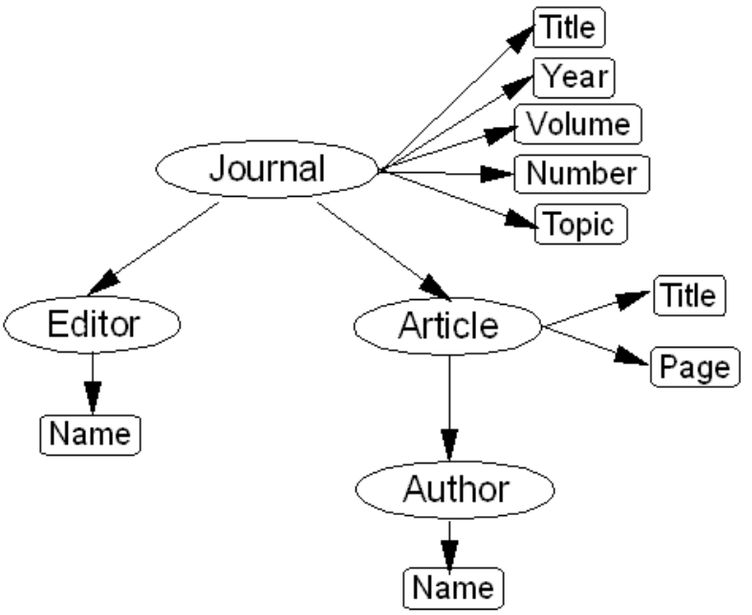


Fig. 3(c). Simplified Schema Graph

4.1.3 Mapping XML Schema Graph into Item Type Definition

Given a simplified XML schema graph, the mapping to a CMS item type is straightforward:

1. Create a root component type **Te** for the root element node **Ne**
2. Create a component type **Te** for each non-root element node **Ne**; the parent component type of **Te** is set to the type of **Ne**'s parent element node; the minimum cardinality and the maximum cardinality of **Te** are set to the values of minOccurs and maxOccurs, respectively, which are annotated on the edge from **Ne**'s parent to **Ne**
3. For each component type **Te**, create a ComponentId that will identify each element instance of the component type
4. For each component type **Te**, create a ParentComponentId that will identify the ComponentId of each parent element instance
5. For each component type **Te** of an element node **Ne**, for each target attribute node **Na** of **Ne**'s outgoing edges, where **Na** points to a primitive type node **Np**, create an attribute for **Te** of data type **p**

As an example, Journal.xsd in Figure 3(a) maps to the Item Type example of Figure 2(a) given in Section 3.2. We take this direct mapping approach, which is similar to the fixed mapping in [2], because we believe that a transformation layer, such as XSLT, can be conveniently added on top of it to perform XML transformations at the XML schema level to achieve a different item type hierarchy if desired.

4.2 XML Document Storage

Given the prior described mapping from an XML schema to an Item Type, the corresponding mapping from an XML document to a CMS Item is straightforward. Given an XML document, our system first parses the document to validate its conformance to the referenced schema. If it does not, the store request is rejected. Otherwise, the document is shredded into "fragments" according to the schema-to-Item-Type mapping. The parent-child relationship between fragments is added into the fragments and then each fragment is inserted in the corresponding component table. As an example, journal1.xml in Figure 4 maps to the item example with ItemId 11 of Figure 2(a) given in Section 3.2.

In addition to storing the values of elements and attributes as an instance of a CMS item, users can optionally store the original XML documents in an Object Server. Thus, no recomposition is required when users retrieve entire XML documents, which preserves the look and feel of the original documents and is a requirement in many business (e.g. legal) applications.

```
<?xml version="1.0" encoding="UTF-8" ?>
<Journal xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
               xsi:noNamespaceSchemaLocation="Journal.xsd"
               Title="IBM Systems Journal" Year="2001"
               Volume="40" Number="4"
               Topic="Knowledge Management">
    <Editor Name="John J. Ritsko" />
    <Editor Name="Alex Birman" />
    <Article Title="Knowledge management technology"
                 Page="814">
        <Author Name="A. D. Marwick" />
   </Article>
    <Article Title="Text analysis and knowledge mining sys-
     tem"
                  Page="967">
        <Author Name="T. Nasukawa" />
        <Author Name="T. Nagano" />
   </Article>
</Journal>
```

Fig. 4. journal1.xml

4.3 Integrated Query

We have implemented query capabilities to support integrated querying of XML and non-XML data. This is achieved by defining an XML view of the Library Server data model. This enables the use of XQuery [21] as the query language. Figure 5 shows the XML view ("cm.xml") of the data in Figure 2. In this XML view, each Item in the Library Server is represented as an XML element which is a child of the document root node. The nesting of the XML elements mimics the hierarchy of components within an Item. The attributes of components are represented by XML attributes of the corresponding XML element. A Link relationship is represented by an <OutboundLink> element nested within the source Item of the Link and also by an <InboundLink> XML element nested within the target Item of the Link. The TargetItemRef attribute of an <OutboundLink> element is an XML IDREF which references the ItemId of the target Item.

Similarly, the SourceItemRef attribute of an <InboundLink> element is an XML IDREF which references the source Item. Representing link relationships as XML elements nested within the XML elements representing the source and the target Items allows most queries to be expressed using just the XPath subset of XQuery.

As explained in the previous sections, we have built the necessary technology that enables the IBM Content Manager for storage of XML documents conforming to a schema. The XML documents are shredded and stored in the Library Server data model. This mapping to the Library Server data model is consistent with the XML view of non-XML data, i.e. the XML view of the Library Server data corresponding to a shredded XML document is consistent with the original XML document. This enables querying XML and non-XML data in the Library Server in a uniform way. It is possible to query both types of data in a single query. A single query processor supports queries against both types of data.

```
<Journal ItemId="11" Title="IBM Systems Journal" Year="2001"
Volume="40" Number="4" Topic="Knowledge Management">
    <Editor Name="John J. Ritsko"/>
    <Editor Name="Alex Birman"/>
    <Article Title="Knowledge management technology"
Page="814">
        <Author Name="A. D. Marwick"/>
    </Article>
    <Article Title="Text analysis and knowledge mining sys-
tem" Page="967">
        <Author Name="T. Nasukawa"/>
        <Author Name="T. Nagano"/>
    </Article>
    <Outbound_Link TargetItemRef="22" LinkType="pdf-content"
LinkItemRef="32"/>
</Journal>
<Journal ItemId="12" Title="IBM Journal of Research and De-
velopment" Year="2001" Volume="45" Number="5" Topic="Advanced
Semiconductor Lithography">
    <Editor Name="George G. Hefferon"/>
   <Article Title="Review of technology for 157-nm lithogra-
phy" Page="605">
        <Author Name="A. K. Bates"/>
        <Author Name="M. Rothschild"/>
    </Article>
    <Article Title="TCAD development for lithography resolu-
tion enhancement" Page="651">
        <Author Name="L. W. Liebmann"/>
    </Article>
    <Outbound_Link TargetItemRef="21" LinkType="pdf-content"
LinkItemRef="31"/>
</Journal>
<Location ItemId="21" FileName="/journals/resanddev.pdf">
    <Inbound_Link SourceItemRef="11" LinkType="pdf-content"
LinkItemRef="31"/>
</Location>
<Location ItemId="22" FileName="/journals/systems.pdf">
    <Inbound_Link SourceItemRef="12" LinkType="pdf-content"
LinkItemRef="32"/>
</Location>
```

Fig. 5. XML View of Library Server metadata

The Library Server data model is implemented using a relational database with SQL as the native query language. To support XQuery, we implemented a query processor to translate XQuery into SQL**.** The query processor uses the traditional Syntax Directed Translation approach to first build an internal tree representation of the input XQuery expression. The query processor then performs a post-order traversal of the tree. The processing for each node generates the SQL needed to evaluate the XQuery subexpression corresponding to the subtree with that node as the root. This SQL is obtained by combining the SQL for the children of the node being processed. The SQL that is generated also depends on the context in which the subexpression appears. Figure 6 shows the algorithm used to generate SQL.

In Figure 6, the function *GetSQL* generates the SQL to evaluate the subtree with *Node* as its root with *Context* as the context. It recursively calls itself on each child of *Node* in order from left to right. The function call *Child*(*Node*, *i*) returns the *i* th child

of *Node* from left. The context in which the SQL for a child is obtained depends upon the type and the context of its parent and the SQL generated for its preceding siblings. The function call *GetContext*(*Context*, *Node*, *SQL*[1..*i*-1], *i*) returns the context for generating SQL for the *i* th child from left. The context is usually an SQL alias name for the database table corresponding to a component within an Item Type. The array *SQL* holds the SQL generated for all the children nodes. *SQL*[*m*..*n*], $m < n$, represents the SQL generated for all the children between *m* and *n*, both inclusive and SQL_i represents the generated SQL for the *i* th child. The SQL for a node is obtained by combining the SQL for all its children using the *CombineSQL*(*Node*, *SQL*) function call. This call simply returns the SQL for *Node* for the case when *Node* is a leaf node. This is where the recursion terminates. The array *SQL* is empty in this case, since there are no children nodes. Combining the SQL is done in a way that is specific to the type of the parent node. The SQL itself is internally represented in a tree form which makes combining easy.

```
GetSQL(Node, Context)
begin
  N = NumberOfChildren(Node);
  for i = 1 to N
  do
    NewContext = GetContext(Context, Node, SQL[1..i-1], i);
    SQLi = GetSQL(Child(Node, i), NewContext);
  done
  return CombineSQL(Node, SQL);
end
```

Fig. 6. SQL Generation Algorithm

4.4 Linking XML Documents

Besides providing an integrated view among XML documents and CMS Items, we also provide ways to tie the two entities together. As described in Section 3.2, links and references are used to tie CMS Items together. Reference is unidirectional which connects from the current location of a component to another Item. A reference is defined as an attribute of a component. Link is bi-directional, i.e. through the query interface, we can find all the inbound and outbound links to a given item.

We extend these two content management features for XML documents such that links can be created among the XML documents. To support this functionality in XML, we use a subset of XLink [22] as the format to represent the linking information.

4.4.1 XLink

XLink is one of the W3C standards which allows elements to be inserted into XML documents to create and describe links between resources. XLink consists of two kinds of links: simple links and extended links. A simple link connects one element in the linking document to a target document. A simple link is unidirectional. It is somewhat similar to the hyperlink of HTML. Figure 7 shows an example which contains a simple XLink. As part of the investment element, it contains a simple link to an investment profile in the location, invest123.xml. A simple link element is indicated by

having an attribute xlink:type with the value “simple”. On the other hand, an extended link element has the value “extended” for the same attribute.

invest123.xml

```
<investment account="123">
<stock>
  <symbol>IBM</symbol>
  <quality>100</quality>
</stock>
<index>
  <symbol>QQQ</symbol>
  <quality>500</quality>
</index>
</stock>
```

profile_jhui.xml

```
<profile xmlns:xlink="http://www.w3.org/1999/xlink">
  <name>Joshua W Hui</name>
  <phone>408-911-2222</phone>
  <investment xlink:type="simple" xlink:href="invest123.xml">
   My Investment
  </investment>
</profile>
```

Fig. 7. A Simple Xlink example

An extended link can connect multiple resources that can be either local (i.e. being a linking element in the current XML document) or remote (i.e. being addressed via a URI reference). An extended link element contains a set of sub-elements that have either “locator”, “resource”, or “arc” as the value of the xlink:type attribute. A locator-type element specifies a remote resource participating in the link. A resource-type element indicates a local resource. An arc-type element describes connections among resources which would be locator-type element or resource-type element. The xlink:label attribute provides the labels for resources. The xlink:to and xlink:from attributes of an arc-type element specify the labels of which resources a connection is established.

Figure 8 shows an example of using an extended XLink to capture the relationship between a talk and a paper. As the example shows, an extended XLink can be used to create links among remote resources. It is not required to define any attribute in either source or target documents. By using resource-type elements, an extended XLink can also be used to create links from local to remote resource. However, in terms of the syntax, it is not as compact as a simple link. Therefore, in general, it is preferable to use the syntax of simple links for a link between a local and a remote resource.

4.4.2 Using XLink to Represent Link and Reference

Due to the directional property and the requirement of defining attributes, we use simple XLinks to represent CMS references and extended XLinks to represent CMS links.

When loading an XML document, an option is given in the interface to indicate whether the system should interpret any XLink element as the link and reference. If the option is on, apart from storing the XLink elements like other XML elements, the system also creates virtual links and references among the resources specified in the XLink elements. With this functionality, users can group XML documents.

```
introXML.xml
<paper title="XML for dummy">
  <section number="1.1"
           title="Introduction of XML">
    This is an introduction of XML.
  </section>
</paper>
allXML.xml
<talk title="All about XML">
  <speaker><name>Parag Tijare</name></speaker>
  <time>13:00</time>
</talk>
workshop.xml
<workshop xmlns:xlink="http://www.w3.org/1999/xlink">
  <session><title>XML</title></session>
  <paperlink xlink:type="extended">
    <paperlocator xlink:type="locator"
                  xlink:href="introXML.xml"
                  xlink:label="paper"/>
    <talklocator xlink:type="locator"
                 xlink:href="allXML.xml"
                 xlink:label="talk" />
    <paper2talk xlink:type="arc"
                xlink:from="paper"
                xlink:to="talk" />
  </paperLink>
</workshop>
```

Fig. 8. An Extended Link example

5 Performance

We have obtained some preliminary measurements on how our system handles the insertion and querying of XML documents. For the insertion measurements, we have focused on two classes of XML documents. The first one has a relatively small data size per element. This captures the scenario where XML is used for exchanging business data. In many cases, such type of data comes from B2B exchanges or from the XMLization of relational database data. The second one has a larger data size per element. It represents applications that use XML for storing both the metadata as well as the content of documents such as journal, magazine, or newspaper publications. For the query measurements, we used the XMark benchmark developed under the XML benchmark project at CWI, Netherlands [17]. Our system is built on top of the IBM Content Manager and uses IBM DB2 UDB V7.2 as the underlying relational database system. The measurements were obtained on an IBM Pentium 4 NetVista workstation with 512 MB running at 1.8 GHz.

Figures 9 and 10 show the document loading time for the two types of XML documents. In both figures, both X and Y axes use logarithmic scales.

Figure 9 shows the time of loading an XML document for the first type of XML document. Each document contains around 20 bytes of data per element. The data set was prepared in the logarithmic increment of file size. As the figure shows, the loading time is linearly proportional to the size of a document as well as the number of elements in the document. Other than the total loading time, we also measure the exe-

cution time of the SQL statements (INSERT and SELECT statements). Our experiment shows that the majority of the time was spent on the SQL INSERT statements. In all our experiments, the number of INSERT statements for each document loading was equal to the number of elements plus three (one for storing the mapping of document name to a global unique id, one for the root element, and the last one for storing the unique id in the CMS catalog).

Similar results were also obtained for the second type of XML document as shown in Figure 10. Each document contains around 240 bytes of data per element. The loading time is also linearly proportional to the size of a document. In this case, the time is much shorter for the same document size (for a 9K document, it is 1 second vs. 2.5 seconds). This is because, for the same document size, the number of SQL inserts is much smaller in this type of document.

In all our experiments, the SQL insert time is always the dominant factor of the document loading time. Each INSERT statement on average takes around 7 msec for the first type of document and around 16 msec for the second type. Even though the execution time for the second type is more than double that of the first type, the insertion is actually more efficient, considering the data size per element is around 12 times larger. This also explains the shorter loading time for the same size of document in the second case. This suggests that one should optimize the number of insertion operations when storing XML documents. One technique would be inlining elements as suggested in [2], [18]. Their goal is to eliminate the number of joins for query processing. In this case, it also helps to reduce the number of SQL insert operations per document. Another technique would be filtering, i.e. only shred the portion of the document which is needed for query purpose. For the rest, it can be stored either in a CLOB column or in a data server (Object Server in our case).

Our experiment shows that, for 9 KB documents, our system can load one type-two XML document per second or about 29 K documents for an eight-hour work day. For type-one, our system can load about 12 K documents per day. For 1.2 MB documents, it can load about one document per minute for type-two or one per 4 minutes for type-one. This demonstrates that our system running on a desktop PC can easily handle the workload of most small and medium size businesses. With a more powerful machine (e.g. a high end SMP), we are confident that it can meet the challenges of most big enterprise workloads.

To get some preliminary information on query performance with our system, we used 9 queries from the XMark benchmark for query performance measurements. We made modifications to these queries, such as translating FLWR expressions to XPath expressions and removing aggregate functions, so that they can be processed by our query processor. We could not get measurements for the other queries in the benchmark because our query processor does not have full support for XQuery yet. Since the set of queries were modified in our experiment, it is not advisable to compare our results with performance numbers from other experiments. We used a scaling factor of 0.25 for the benchmark. This translates to an XML document of size 25 MB. Table 2 shows the time for executing these queries. This measurement does not include the time taken for reconstruction of XML results from SQL tuples.

Our experiment shows that the time for translating XQuery to SQL is negligible in all cases. Majority of time is spent in executing the generated SQL statement.

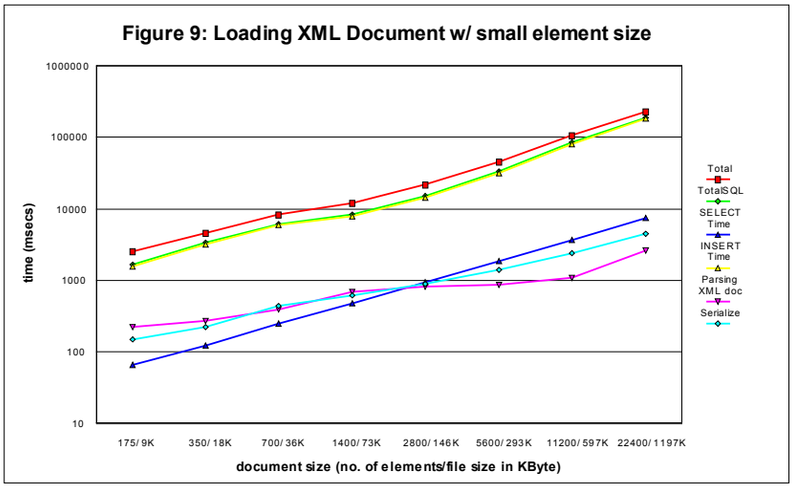


Fig. 9. Loading XML Document w/small element size

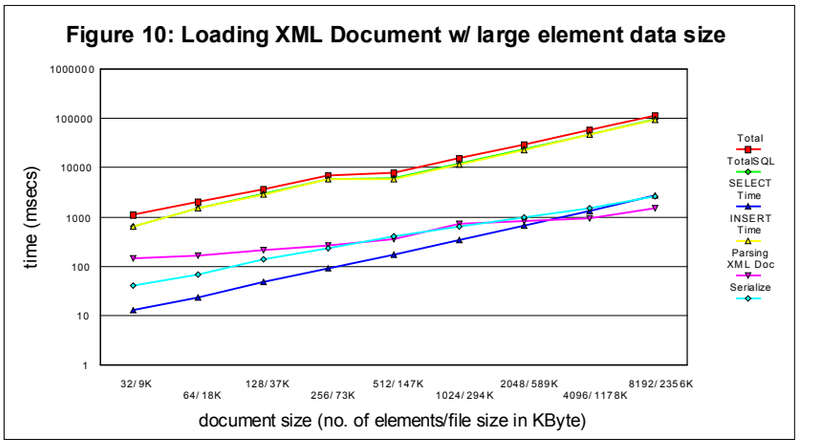


Fig. 10. Loading XML Document w/large element data size

Table 2. Test queries

Query Number	Time (milliseconds)
Query 1	185
Query 5	150
Query 6	225.5
Query 7	190
Query 8	230
Query 9	555.5
Query 11	1186.5
Query 17	190.5
Query 19	246

6 Conclusions

Many research projects have focused on providing efficient storage or mapping models for XML repositories. Our focus in this paper, however, is on how to integrate XML document with other kinds of content, for the following two main reasons:

1. Other than XML documents, businesses will continue to have other kinds of data like video, audio, image and text documents that need to be managed.
2. Often XML is used as the representation for describing metadata of other types of content. Therefore, both the XML document and the content should be managed in an integrated fashion.

Since a content management system has been developed for traditional document management, we have find it to be a suitable platform for building an integrated XML document management system. In our approach, we define a direct mapping from an XML schema to a hierarchical data model of a content management system, where several simplifications and transformations are applied first, then a schema graph is generated and simplified, and finally an Item Type is created in the content management system. XML documents are stored as Items of the type. By defining an XML view of a CMS data model, our system provides an integrated query capability across both XML and CMS data via XQuery.

Our preliminary performance measurement shows that our system scales linearly with respect to the XML document size as well as the number of elements in a document. On a desktop PC machine, our system can load one XML document of 9 KB per second or 1 per minute of 1.2 MB, which should be more than sufficient for most small and medium sized businesses. Our preliminary query performance is promising. And for future work, we are planning to measure the integrated query and retrieval performance of our system. Also, we will explore other CMS features like versioning, workflow management, caching and replication, and investigate how they can be tailored to take advantage of the structures of XML documents.

Acknowledgement

The authors would like to thank Matthew Laue for helping in the design discussion of the XML document management system. Also, the authors are grateful to the IBM Content Manager development team. Their dedication and contribution in building the Content Manager product has made this paper possible.

References

[1] Berson, S., Ghandeharizadeh, S., Muntz, R. R., Ju, X.: Staggered Striping in Multimedia Information Systems. In: Proceedings of the ACM SIGMOD International Conference on Management of Data, Minneapolis, Minnesota, May 1994, pp. 79-90.

[2] Bohannon, P., Freire, J., Roy, P., Simeon, J.: From XML Schema to Relations: A Cost-Based Approach to XML Storage. In: Proceedings of the 18th International Conference on Data Engineering, San Jose, California, February 2002.

[3] Dashti, A. E., Ghandeharizadeh, S.: On Configuring Hierarchical Storage Structures. In: Proceedings of the Joint NASA/IEEE Mass Storage Conference, College Park, Maryland, March 1998.

[4] Deutsch, A., Fernandez, M. F., Suciu, D.: Storing Semistructured Data with STORED. In: Proceedings of the ACM SIGMOD International Conference on Management of Data, Philadelphia, Pennsylvania, May 1999, pp. 431-442.

[5] Documentum 4i XML Capabilities. http://www.documentum.com/.

[6] Faloutsos, C., Barber, R., Flickner, M., Hafner, J., Niblack, W., Petkovic, D., Equitz, W.: Efficient and Effective Querying by Image Content. In: Journal of Intelligent Information Systems 3(3/4), pp. 231-262, 1994.

[7] Federighi, C., Rowe, L. A.: A Distributed Hierarchical Storage Manager for a Video-on-Demand System. In: Proceedings of the IS&T/SPIE Symposium on Electronic Imaging Science and Technology, February 1994, pp. 185-197.

[8] FileNet Panagon Products. http://www.filenet.com/.

[9] Florescu, D., Kossmann, D.: Storing and Querying XML Data using an RDBMS. In: IEEE Data Engineering Bulletin, 22(3), pp. 27-34, 1999.

[10] IBM Content Manager. http://www.ibm.com/software/data/cm/.

[11] iManage Solutions. http://www.imanage.com/products/index.html.

[12] Matinez, J. M.: Overview of the MPEG-7 Standard (version 6.0). ISO/IEC JTC1/SC29/WG11 N4509, December 2001.

[13] Microsoft Content Management Server. http://www.microsoft.com/cmserver/.

[14] Naughton, J., DeWitt, D., Maier, D. et al.: The Niagara Internet Query System. In: IEEE Data Engineering Bulletin 24(2), pp. 27-33, 2001.

[15] Oracle XML DB. http://technet.oracle.com/tech/xml/xmldb/content.html.

[16] Oracle9i Database. http://www.oracle.com/products/oracle9i/content.html.

[17] Schmidt, A. R., Waas, F., Kersten, M. L., Florescu, D., Manolescu, I., Carey, M. J., Busse, R.: The XML Benchmark Project. http://monetdb.cwi.nl/xml/.

[18] Shanmugasundaram, J., Tufte, K., He, G., Zhang, C., DeWitt, D., Naughton, J.: Relational Databases for Querying XML Documents: Limitations and Opportunities. In: Proceedings of the 25th Very Large Data Bases Conference, Edinburgh, Scotland, September 1999, pp. 302-314.
[19] Tamino XML Server. http://www.softwareag.com/tamino/.
[20] World Wide Web Consortium. Extensible Markup Language (XML). W3C Recommendation, February 1998.
[21] World Wide Web Consortium. XQuery 1.0: An XML Query Language. W3C Working Draft, December 2001.
[22] World Wide Web Consortium. XML Linking Language (XLink) Version 1.0. W3C Recommendation, June 2001.
[23] World Wide Web Consortium. XML Schema. W3C Recommendation, May 2001.

Integration of XML Data

Deise de Brum Saccol[1] and Carlos Alberto Heuser[2]

[1]Centro Universitário Luterano de Palmas
Av. Teotônio Segurado, 1501 SUL, Palmas, TO, Brazil
deise@ulbra-to.br
[2]Universidade Federal do Rio Grande do Sul
Av. Bento Gonçalves, 9500, Bloco IV, Porto Alegre, RS, Brazil
heuser@inf.ufrgs.br

Abstract. Various XML instances from different data sources can model the same object of the real world. Query processing or view definition over these sources demands instance integration. In this context, integration means to identify which data instances represent the same object of the real world, as well as to solve ambiguities of representation of this object. The entity identification problem in XML is more complex than in structured databases. XML data, as originally considered, necessarily do not have the identification notion of primary key or object identifier. Thus, it is necessary the adoption of a mechanism that identifies the instances at the moment of data integration. This paper presents a proposal for identifiers attribution to XML instances, based on the use of *Skolem* functions and *XPath* recommendation, as proposed by W3C.

1 Introduction

The available amount of electronic data has grown so fast in the last years. These data come in several formats, from completely unstructured file systems to structured information stored in relational database systems.
In order to integrate data from different sources, it is necessary to identify data instances representing the same object of the real world. The bibliography refers to this problem by the terms *object identification, instance identification or entity identification* [2].

However, not all the applications can be satisfied with the relational, object-relational or object-oriented models; examples are found in the WEB, for which the scheme at the moment they are created is not known. To support the integration of these types of applications, models of semistructured data have been considered. XML is being recognized as an important approach to represent semistructured data. XML data, as originally considered, do not demand identifiers. Even when they exist, the identifiers restrict to distinguish the instances inside a document, not being useful to the integration between different documents. At the same time, the lack of structure in these data makes impracticable many of the techniques used in the

S. Bressan et al. (Eds.): EEXTT and DIWeb 2002, LNCS 2590, pp. 68-80, 2003.

integration of instances in structured data sources. Trying to decrease this deficiency, this paper presents a proposal of an approach of identification of XML instances.

The proposal of identification of XML instances presented in this paper is part of a Master Thesis developed at Federal University of Rio Grande do Sul [19], which focus on the access and integration of heterogeneous sources. XML data are extracted from the sources, integrated and materialized in a relational database. End user queries are posed directly over this database, using a language based on SQL.

For the materialization of XML data, two main activities must be carried through:

- the generation of the relational logical scheme (based on an ontology that describes the problem domain of the sources);
- the instantiation and incremental maintenance of this database, in order to keep the data consistency in relation to the sources.

The materialization module is showed in Fig.1.

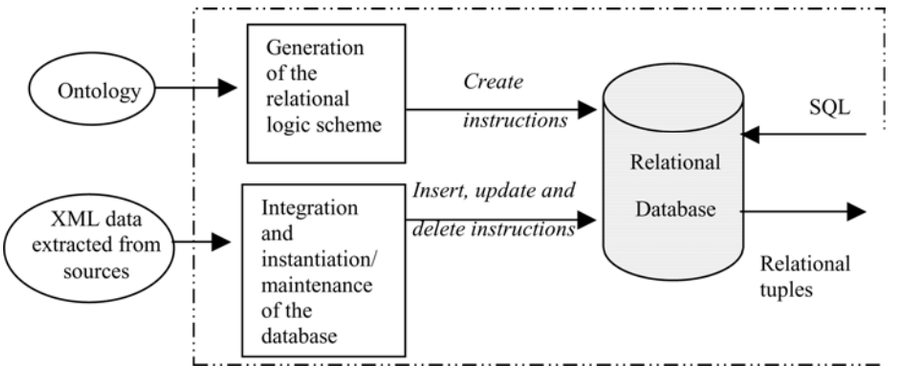


Fig.1. Materialization Module

The problem of instance identification arises at the moment of integration of the extracted data from the sources. This problem has been studied in the heterogeneous database area, which presents some proposals for solving it. Integration of XML data is a recent research area and here is the focus of this paper.

The present paper is organized as it follows. Section 2 presents the State of the Art about instance identification in heterogeneous database systems, along with the approach here proposed. Section 3 presents four approaches used on the properties values conflict resolution of a real world object, as well as the proposal adopted here.

2 Instance Identification

In this section, the main approaches used in heterogeneous databases to solve the problem of instance identification are pointed, as well as presented the approach proposed in this work.

2.1 State of the Art

Universal Key This is the simplest method for data integration. It is based on the existence of a common key between the instances to be integrated [2, 14, 16, 17, 23]. However, this approach is restricted, since the sources not always have a common key, as XML data.

Key Equivalence Specified by the User This approach requires that the user specifies equivalence between the instances, for example, using a mapping table of the local identifiers from each source to the global identifiers in the integrated system. This technique is used in [1, 18, 20]. The disadvantage of this technique is that the mapping table can be considerable and present difficult maintenance, handled by the database administrator, not in a automatic way.

Attributes Equivalence Not having a common key to integrate the instances, some proposals suggest using common attributes. For each pair of records from two sources, a matching value can be processed, which measures the degree of similarity between two instances. All the common attributes or only some of them (specified by the integrated system administrator) can be used to determine the object equivalence [4, 15, 22, 24].

Queries Defined for the User A query can be defined for each type of object of the real world, in order to construct a unique identification property for objects of this type. [8] uses this technique. Always when two instances of the same type return the same result for the query, they are considered the same object of the real world.

Inference Rules Lim [13] presents one technique for derivation of absent identifiers based on inference rules; the proposal uses extra semantic information to automate the process of identification between relations that do not share a common key. It considers the use of an extended key to identify instances, which can be defined as the union of keys of the participant data sources.

Boolean Functions Defined for the Designer When a common key between the data to be integrated does not exist, and when attributes in common that can be used in the identification of the instances do not exist, the designer can define a function for such intention. An example of this technique is presented in [24], which uses a function *close_names()*; this function returns a boolean value, depending on the similarity between two strings characters passed as arguments. This function could, for example, ignore names abbreviations in the process of instances identification, and consider that "Marcos Santos" and "Marcos A. Santos" refers to the same object of the real world.

The proposals found in literature for integration of XML data usually presume that the elements already have identifiers. The proposal of this paper is based on OWA (Open World Assumption) approach, since it aims to integrate data from the WEB. The OWA allows adding objects and improving knowledge of objects already stored. That is, the integrated system is open for new objects from WEB. In this way, the techniques based on the existence of an identifier in the data source (*Key Equivalence Specified for the User, Inference Rules*) are also excluded.

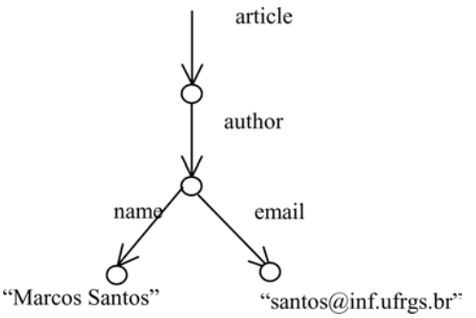
The technique *Attributes Equivalence* is applicable to the semistructured data, but it was not used in this paper for demanding great intervention of the user in the specification of the similarity degree of values.

As presented in section 2, one of the techniques for instance identification avaiable in literature is the submission of queries over these data, in order to return the value of one or either a set of properties that uniquely identify it [8]. In this way, the proposed approach in this article aims to submit a query to the data sources, which returns the identifier of the instances. This means that two instances that model different objects of the real world never return the same result for this query. The technique here proposed combines the techniques *Queries Defined for the User* and *Boolean Functions Defined for the Designer.*

2.2 Technique Proposal for Identifying XML Instances

The data model adopted consists of a tree, representing the XML document, composed for nodes and edge. Nodes represent objects of the document, while the labels are represented in the edges. Complex objects point out other objects (for instance: object *author* in the tree below); atomic objects contain values of a basic type, represented for a string of bits (for instance: object *name* in the tree below). For example, the following XML document is represented in the tree below:

```
<article><author><name>Marcos Santos </name>
<email>santos@inf.ufrgs.br</email></author></article>
```



In object-oriented data [10], as well as in semistructured data (XML-QL [7]), the notion of instance identification has appeared in the form of *Skolem* functions. An alternative approach is the use of an *id function*, as introduced in [11].

The term *Skolem function* comes from the Logic of Predicates [9] and is used to assign a function that, from a given object, constructs its identifier.

XML-QL uses this type of function to transform data, for example, from a DTD to another one [7]. To illustrate this, consider the following part of a DTD that defines a type *author*:

```
<!ELEMENT author (firstname, lastname)>
```

One can write a XML-QL query that transforms data from this DTD into data that are in accordance with another DTD, for example:

```
<!ELEMENT person (lastname, firstname, address?,phone?,
publicationtitle*)>
<ATTLIST person ID ID #REQUIRED>
```

The query below extracts authors and transforms them into elements <person>.

```
WHERE
<$><author><firstname>$fn</><lastname>$ln</></></>
IN "www.a .b.c/bib.xml",
CONSTRUCT  <person ID=PersonID ($fn, $ln)
<firstname>$fn</> <lastname>$ln</> </>
```

Always when an element <person> is produced, its ID becomes PersonID ($fn, $ln). PersonID is a *Skolem* function, and its intention is to generate a new identifier for each distinct value of firstname and lastname. A more formal description of *Skolem* Functions can be found in [9]. XML-QL does not specify as the *Skolem* functions are implemented.

The mechanism here considered consists:

- to use *Skolem* functions to identify objects;
- to implement the *Skolem* functions through queries in a XML language, specifically *XPath* [3].

It follows some examples of *Skolem* functions codified in *XPath*. It assumes that the first and last names of an author are enough to uniquely identify a data instance of this type. Thus, the query over this element must return these two strings of characters in order to identify them. Always when the same values are returned from the query, it will be considered that they are the same object of the real world; different results mean that the two data instances model two distinct objects.

The *Skolem* function could, for example, concatenate the two strings of characters, convert them in capital letter, eliminate the white spaces (if they exist), map them in a numerical value etc; the final result would be the identifier of this instance (ID).

Considering the same case and assuming two instances of XML data, from different sources, with distinct structures:

```
<person><lastname>Santos</lastname>
<firstname>Marcos</firstname></person>
```

```
<author><firstname>Marcos</firstname><middlename>Albert
</middlename><lastname>Santos</lastname></author>
```

Considering that the information contained in the labels <firstname> and <lastname> is enough to identify the instance; the idea is to submit one query to each data source in order to return the content from these labels, concatenate the two strings of characters and convert them in capital letter, generating the following identification: ID = MARCOSSANTOS. With this identifier, one can have the knowledge that both XML instances refers to the same object of the real world. So, this instance will be considered just once, at the moment of the data integration.

The example above was simplified for understandable needs. The necessary information for identification was represented directly in the labels <firstname> and <lastname>. A simple query in a XML language would return the expected values. However, when refering to semistructured data, this it is not always correct. The

necessary information for instance identification could be found inside a string of characters of one specific label; in this case, it is necessary a processing in order to extract it before the generation of the identifier. In another case, it could be necessary to extract subparts of some labels of the XML document, concatenate them and convert them in a specific format. For example, one could consider that a scientific event is identified by its name and the year of accomplishment. Such information is available in the following instance XML:

```
<article><author><name>Marcos Alberto Santos</name>
<email>santos@inf.ufrgs.br</email></author>

<title>Caching XML Data</title>

<event>Brazilian Simposium                              on
Databases, RJ, july  2001</event></article>
```

To generate the identifier of this event, it is necessary to extract the name of the event and the year of its accomplishment. Although this information is visible to a person, it is not represented in a directly way for a XML query language. The name and the year of the event are inside the label <event>, and must be extracted in order to generate the identifier of the instance.

To reach this goal, the *XPath* recommendation is used in the article, considered by W3C. *XPath* it is a language that allows to refer to parts of a XML document and to supply basic facilities to handle strings of characters, boolean and numbers [3]. This recommendation models a XML document as a tree of nodes. The basic syntactic constructor in *XPath* is the *expression,* a string of characters that consists of instructions to select an element, attribute, other markup structures, or a string of text [3]. They identify an item for its location in the hierarchic structure of the document. For example, the expression *author* selects children of the current element that has the name *author*. For current element, it is meant the current position in the tree of XML document. Another example is the expression *book/title*, which selects the title of an element *book*.

After evaluating an expression, the result is a set of selected nodes. Expressions can contain, recursively, other expressions used to filter sets of nodes, through the use of predicates: author[email=santos@inf.ufrgs.br]. The evaluation of the expression above returns the author whose email is the one specified in the string of characters.

Moreover, the *XPath* recommendation propose a varied set of functions for the handling of position elements tests (select the first author), values of attributes (select the attribute *edition* of the element *book*), manipulation of strings of characters (select the paragraph that starts with *'XML is a standard'*). A full description of *XPath* can be found in [3]. Basically, the proposal of the paper uses expressions and functions that make possible the manipulation of strings of characters: extraction of substrings of characters contained in elements, concatenation, conversion for upper/lower case, elimination of white spaces etc.

Returning to the example about the identification of an event for its name and year from accomplishment. The name of the event is available from the beginning of the label and extends to the first comma. Assuming that this is a regular standard in all the labels <event> of a specific XML document. One can take advantage of this fact, and using *XPath*, to extract the information of the event name, using the following function*: string substring_before(string, string).* This function returns the substring of

the first argument that precedes the first occurrence of the second argument in the first argument. For example: *substring_before("Brazilian Simposium on Databases, RJ, july 2001", ",")* returns "*Brazilian Simposium on Databases*" (the several white spaces are on purpose). The year of the event also is necessary for identification; assume that this is always available in the four last positions of the label <event>. To extract it, one can use the function

substring("Brazilian Simposium on Databases, RJ, july 2001", string_length("Brazilian Simposium on Databases, RJ, july 2001")-3,4).

This function returns the substring from the first argument, starting in the position specified in the second argument with the size specified in the third argument. Assuming that the function *string_length* returns the number of characters in the string passed as parameter, the expression above returns *2001*. (for optimizations about the use of these functions, see [3]). Now, making use of the name of the event and its year of accomplishment, the identification of the instance could be generated. The *Skolem* function can carry through a processing on the name and the year of the event. The type of this processing is defined by the integrated system administrator. Consider that the generated identifier is a string of characters, resulted of the concatenation of the two arguments, previously converted to capital letter and without the presence of multiple white spaces between the words. Using the *XPath* functions, the steps are shown below. The partial results are attributed to temporary variables for better understanding:

1) eliminating white spaces: *a = normalize-space (*"Brazilian Simposium on Databases*")*

a = "Brazilian Simposium on Databases"

2) converting to capital letter:

b = translate (a, "abcdefghijklmnopqrstuvwxyz", "ABCDEFGHIJKLMNOPQRSTUVWXYZ")

b = "BRAZILIAN SIMPOSIUM ON DATABASES"

The generated final identifier will be the result of the concatenation of the string of characters "b" with the year of the event: *ID = concat (b, " ", "2001")*

ID = "BRAZILIAN SIMPOSIUM ON DATABASES 2001"

Putting all the steps together, the responsible function for the identifier generation of an event in a specific XML source that follows the standards above would have the following style:

ID = (concat(translate(normalize-space(substring_before(event, ","))), "abcdefghijklmnopqrstuvwxyz", "ABCDEFGHIJKLMNOPQRSTUVWXYZ"), " ", substring(event, string_length(event)-3,4))

ID = "BRAZILIAN SIMPOSIUM ON DATABASE 2001"

It must be pointed out that the *XPath* language includes an ample variety of functions that allow the manipulation of XML documents. This ample flexibility

allows the most varied handlings of XML instances, and the generated identifier can be resulted from complex processings in the structure of the document.

In the example above, one assumed that an event was identified for its name and year of accomplishment. Such assumption comes upon a study of the problem domain, and it is responsibility of the system designer that integrates the sources. This external knowledge of the domain is essential to make the instance identification.

As already stated, XML data are semistructured data. The necessary information for identification of one specific object (for example, event) can be settled in several ways when we go from one source to the others. In this way, a different *Skolem* function must be written to each source when the structure is not the same one. For example, assuming that in another XML document, the name of the event and its year of accomplishment were stored in separated labels, as shown below:

```
<event><nameEvent>Brazilian Simposium          on
Databases</nameEvent><yearEvent>2001</yearEvent></event>
```

The *Skolem* function for generation of the identifier of event for this source would be of the style:

ID = (concat(translate(nameEvent, "abcdefghijklmnopqrstuvwxyz",

"ABCDEFGHIJKLMNOPQRSTUVWXYZ")), " ", yearEvent)

ID = "BRAZILIAN SIMPOSIUM ON DATABASES 2001"

It notices that, despite the structure of the XML document vary between the sources, it is assumed that inside of a source it remains regular. In the example above, it is considered that the necessary information for the generation of the identifier was available in the labels <nameEvent> and <yearEvent>. If this rule is not satisfied in the entire XML document, the generated identifier will not be the expected one.

From the considerations above, it can be evidenced that the integrated system administrator is responsible for the definition of which information is necessary for identification of the entities to be integrated. Moreover, a *Skolem* function must be written to each XML source (assuming that its structures, labels and disposals of the necessary information are not the same ones) for each type of object to be identified; all these functions must return the same identification, when refering to the same entity of the real world.

3 Attributes Values Conflict Resolution

After two instances were identified as the same object of the real world, the values of its properties can present conflicting values, for example, two different addresses for the same author. By properties, we mean the attributes of an object. In this paper it is considered that object properties are represented in XML for atomic elements or attributes, or either, for leaves of the representative tree of an XML instance. For the handling of this type of conflict, some approaches had been proposed in available literature for structured databases. Along with the proposed approach for XML, these techniques are presented below

3.1 Conflict Resolution State of the Art

Aggregation Functions In this approach, the aggregation functions of the query languages are used (minimum, maximum, average, etc.) to solve conflicts in attributes values. For instance, considering the case that the value of the price of a book is represented with different values in two data sources; it could be used a minimum function and be considered only the minor value for the property *price* in the integrated system. This technique is proposed in [2, 5, 15, 18]. The disadvantage of this technique is that the used functions can restrict the type of attribute to which can be applied; as example, a minimum function can only be applied to numerical values.

Partial Values When different values of a property cannot be mapped to only one value in the integrated system, a partial value can be generated. This partial value is a set of values, from which only one is considered correct [6]. This only value is resulted of the intersection of some conflicting values in this property. For example, considering that three XML instances model the same object *author* and present the following values for the property address: address 1 = "X Street, 110, Bananeiras District", address 2 = "X Street, 110, Downtown" and address 3 = "X Street, 110". The value considered correct in the integrated system would be the result of the intersection between the conflicting values, or either, "X Street, 110".

Probabilistic Partial Values In this proposal, probabilities are attributed to the partial values. Extended selection operations can filter tuples that do not satisfy the query condition with the expected probability [21]. [14] proposes an extension to the relational model to store these conflicting values of the attributes. For example, considering the previous case, some method of attribution of probabilities to the conflicting values of the property *address* can be applied: address 1 = probability 0.66; address 2 = probability 0,33. A query specifying an expected minimum probability of 0.5 would return only the value from address 1 to the user.

Storage of All Attributes Values [15] considers a global object model that stores all the conflicting values of the attributes found in the data sources; the model allows user queries on attributes with the original values of the sources and with the solved values in the integrated model (for example, using aggregation functions for the resolution, whenever it is possible). Differently of the previous proposal, this technique does not assign probabilities to the conflicting values of the attributes; it allows to query all the values found in the sources for the same property of a real world object, as well as the solved value in the integrated system, whenever it is possible.

Analyzing these techniques, it is noticed that they do not have a generalized application, depending on the application and the property that is being considered.

3.2 Technique Proposal for Attributes Values Conflict Resolution

In this paper, we looked for a generic and applicable solution for data sources from WEB. This proposal joins two ideas, normalization of values and time stamps.

For *normalization of values,* we mean the transformation of the original values from the sources in a normalized value; this normalized value can be compared with the value from another source.

For instance, considering two data sources. The first one stores the information about 'days of the week' in Portuguese language format. The second one stores them in numerical format. To normalize the values, data from these ources could be converted to a canonic set of values. In this example, the values from both sources could be transformed in English language format. We propose to normalize these values using XSLT language.

Moreover, it is assumed that each data source has a time stamp which identifies its last update. The proposed mechanism consists, in case of conflict of the normalized value, to use the value that has the most recent time stamp.

Assuming that the author identified by "MARCOSSANTOS" is represented in two data sources, as shown below:

```
<name>Marcos A. Santos</name><address>X
Street,110</address>

<firstname>Marcos</firstname><lastname>Santos
</lastname><address>Y Street, 200</address>
```

Considering that source1.xml had the last update (time stamp) on January, 1st 2000 and the source 2.xml on July, 3rd 2000. Thus, the responsible module for the source integration verifies the time stamps of the sources that refer to this author and considers the address "Y Street, 200" as the correct one, since the source 2 is the one that was modified latest. The information about time stamps and about which sources refer to which integrated instances is necessary to adopt this technique.

Let's take a look at another example concerning to the conflict resolution of value of properties:

```
<title>Caching XML Data</title><edition>2</edition>
<title @edition = "2ª" Caching XML Data</title>
```

The property *edition* of the book is represented with different values in two data sources; in this case, a normalized value can be generated in order to match them. For example, we can use the function *translate (edition, "1234567890ª", "1234567890")* to get the value "2" of the property *edition*, which is represented in source2.xml for "2ª". The function *translate* returns the first argument with occurrences from characters in the second argument substituted by the characters (in the corresponding position) from the third argument. If there is a character in the second argument without character (in the corresponding position) in the third argument, then the occurrences of this character in the first argument are removed. Thus, the two labels reflect the same value (2); only the normalized value is returned from XML sources and this value is used for matching the value of the property.

4 Final Considerations

XML data integration is a new research area. However, the problem of integration of heterogeneous data sources has been extensively studied in the latest years.Taking advantage of that, this proposal for XML instance identification is based on a deep study of the alternatives already known for heterogeneous database systems, as seen in sections 2.1 and 3.1.

The proposed approach seems to be flexible:

1) Objects, represented for complex objects of a XML document (for example *author*, in the examples above) are identified by *Skolem* functions, codified as *XPath* queries to the original data sources.
2) Object properties, represented for atomic elements and attributes of XML, (for example, the book edition) are identified by its content, by default. However, a normalized value can be used as identification, using XSLT language. In the occurrence of conflicts in the normalized values, it is used information from the source that has the most recent time stamp.

At this moment of development, it is demanded the knowledge and the use of XSLT/*XPath*. To make the proposal usable for users who do not dominate these techniques, we are implementing a graphical interface, in which the user can choose between preconstructed skeletons of *XPath* queries. These skeletons can be defined from a grouping of the types of identifications that are used more frequently in XML sources. Examples could be the content of a property, the concatenation of the content of some properties etc.

We have used the Xalan processor for executing the *XPath* queries. Xalan-Java is an XSLT processor for transforming XML documents into HTML, text, or other XML document types. It implements the W3C Recommendations for XSL Transformations (XSLT) and the XML Path Language (*XPath*).

References

[1] Ahmed, R. et al. The Pegasus Heterogeneous Multidatabase System. Computer, New York, v.24, n.12, p. 19-27, Dec. 1991.

[2] Albert, J. Data Integration in the Rodin Multidatabase System. In: International Conference On Cooperative Information Systems, 1., 1996. Papers. [S.l.:s.n.], 1996. p. 48-57.

[3] Bradley, Neil. The XML Companion. 2nd ed. Harlow: Addison-Wesley, 2000.

[4] Chatterjee, A. et al. Data Manipulation in Heterogeneous Databases. Sigmod Record, New York, v.2, n.4, p. 64-68, Dec. 1991.

[5] Dayal, U. Processing queries over generalized hierarchies in a multidatabase system. In: International Conference On Very Large Data Bases, 9., 1983, Florence, IT. Proceedings... Florence: VLDB Endowment, 1983. p. 342-353.

[6] Demichiel, L. G. Resolving Database Incompatibility: An Approach to Performing Relational Operations over Mismatched Domains. IEEE Transactions on Knowledge and Data Engineering, New York, v.1, n.2, p. 485-493, Dec. 1989.

[7] Deutsch, A. et al. A query language for XML. Journal WWW8/ Computer Networks, [S.l.], v.31, n.11-16, p.1155-1169. Avaiable at: <http://www.research.att.com/~mff/files/final.html>. Access: Apr. 24 th, 2002.

[8] Gogolla, M. Identifying Objects by Declarative Queries. In: Chomicki, Jan; Saake, Gunter; Sernadas, Christina. The Role of Logics in Information Systems. [S.l.:s.n.], 1995. (Dagstuhl-Seminar-Report, n. 121).

[9] Hein, J. Discrete Structures, Logic and Computability. [S.l.]: Jones&Bartlett Publishers, 1995. Preliminary Edition.

[10] Hull, R. et al. ILOG: Declarative Creation and Manipulation of Object Identifiers. In: International Conference On Very Large Data Bases, 6., 1990, Brisbane, AU. Proceedings... Brisbane: VLDB Endowment, 1990. p. 455-468.

[11] Kifer, M. et al. Querying Object-Oriented Databases. In: International Conference On Management Of Data, 1992, San Diego, California, USA. Proceedings... San Diego: ACM Sigmod, 1992, p. 393-402.

[12] Liefke, H. et al. Efficient View Maintenance in XML Data Warehouses. [S.l.]: Department of Computer and Information Science, University of Pennsylvania. Available at: <http://www.cis.upenn.edu/~liefke/papers/whax.ps.gz>. Access: Jan. 20 th, 2002.

[13] Lim, E. et al. Entity identification in database integration. In: International Conference On Data Engineering, 9., 1993, Viena, AU. Proceedings... Viena: [s.n.], 1993. p.294-301.

[14] Lim, E. et al. Resolving attribute incompatibility in database integration: An evidential reasoning approach. In: International Conference On Data Engineering, 10., 1994, Houston, US. Proceedings... Houston: [s.n.], 1994. p.154-163.

[15] Lim, E. et al. A Global Object Model for Accommodating Instance Heterougeneities. In: International Conference On Conceptual Modelling, 17., 1998, Singapore. Proceedings... Singapore: [s.n], 1998. p. 435-448.

[16] Manolescu, I. et al. Agora: Living with XML and Relational. In: International Conference On Very Large Data Bases, 26., 2000, Cairo, EG. Proceedings... Cairo: VLDB Endowment, 2000. p. 623-626.

[17] Papakonstantinou, Y. et al. Object Fusion in Mediator Systems. In: International Conference On Very Large Data Bases, 22., 1996, Bombay. Proceedings... Bombay: VLDB Endowment, 1996. p. 413-424.

[18] Reddy, M. P. et al. A Methodology for Integration of Heterogeneous Databases. IEEE Transactions on Knowledge and Data Engineering, New York, v.6, n.6, p. 920-933, Dec. 1994.

[19] Saccol, D. B. Materialização de Visões XML. 2001. Master Thesis (Master Course in Computer Science) – Instituto de Informática, Universidade Federal do Rio Grande do Sul, Porto Alegre.

[20] Shoens, K. A. et al. The Rufus System: Information Organization for Semi-Structured Data. In: International Conference On Very Large Data Bases, 19., 1993, Dublin, IR. Proceedings... Dublin: VLDB Endowment, 1993. p. 97-107.

[21] Tseng, F. S. et al. A Probabilistic Approach to Query Processing in Heterogeneous Database Systems. In: International Workshop On Research Issues On Data Engineering: Transaction And Query Processing, 2., 1992, Tempe, US. Proceedings... Tempe: [s.n.], 1992. p. 176-183.

[22] Wang, Y.R. et al. The Inter-Database Instance Identification Problem in Integrating Autonomous Systems. In: International Conference on Data Engineering, 5, 1989, Los Angeles, US, Proceedings... Los Angeles [s.n], 1989, p. 46-55

[23] Wiener, J.L. et al. The WHIPS prototype for Data Warehouse Creation and Maintenance. In: International Conference on Data Engineering, 13, 1997, Birmingham, UK. Proceedings...

[24] Zhou, G. et al. Using Object Matching and Materialization to Integrate Heterogeneous Databases. In: International Conference on Cooperative Information Systems, 3, 1995, Viena, AU. Proceedings... 1995. p. 4-18.

XML to Relational Conversion Using Theory of Regular Tree Grammars

Murali Mani* and Dongwon Lee

Department of Computer Science, University of California, Los Angeles
Los Angeles, CA 90095, USA
{mani,dongwon}@cs.ucla.edu

Abstract. In this paper, we study the different steps of translation from XML to relational models, while maintaining semantic constraints. Our work is based on the theory of regular tree grammars, which provides a useful formal framework for understanding various aspects of XML schema languages. We first study two normal form representations for regular tree grammars. The first normal form representation, called NF1, is used in the two scenarios: (a) Several document validation algorithms use the NF1 representation as the first step in the validation process for efficiency reasons, and (b) NF1 representation can be used to check whether a given schema satisfies the structural constraints imposed by the schema language. The second normal form representation, called NF2, forms the basis for conversion of a set of type definitions in a schema language L_1 that supports union types (e.g., XML-Schema), to a schema language L_2 that does not support union types (e.g., SQL), and is used as the first step in our XML to relational conversion algorithm.

1 Introduction

The theory of regular tree grammars [16, 6] provides an excellent framework for understanding various aspects of XML schema[1] languages [14], and have been actively used in many applications including XML document processing (e.g., XQuery from W3C [4] and XDuce [9]) and XML document validation algorithms (e.g., RELAX, TREX, and RELAX-NG [5]). It is also used for analyzing the expressive power and closure properties of the different schema language proposals [14].

Our emphasis, in this paper, is on understanding the data modeling aspects of XML schemas. XML schema provides several unique data modeling features, e.g., union types, that are not present in traditional database models such as the relational model, and this makes this study challenging. Several problems have been studied regarding the data modeling aspects of XML schemas. The foundations of the current work are based on our earlier work [13]. Here we describe

* This author is partially supported by NSF grants 00861 16, 0085773, 9817773.

[1] We differentiate two terms – XML schema(s) and XML-Schema. The former is a general term for schema in XML model while the latter refers in particular to the XML schema language proposed by W3C [17].

S. Bressan et al. (Eds.): EEXTT and DIWeb 2002, LNCS 2590, pp. 81–103, 2003.

how entities and relationships, which form the basis of data modeling, can be represented using the features provided by XML, such as elements, attributes, parent-child relationships, ID-IDREF attributes, and using inclusion dependencies. We further describe how the various definitions in a given XML schema can be mapped to entities and relationships. For example, a parent-child relationship from book to author represented as book → author* in XML can be mapped to an ordered 1:many relationship, where for every book, we have an ordered list of authors.

A related area that has generated a lot of interest is mapping from relational to XML models [11, 12]. In [12], we represent the relationships expressed using inclusion dependencies in the relational model as parent-child relationships, as ID-IDREF attributes, and sometimes as inclusion dependencies in the XML model. For example, if we have a relation book and a relation author, and author has a foreign key referencing book, then we can represent it in XML as book → author*. In this paper, we focus on the reverse conversion from XML to relational schemas. This is necessary for storing XML documents using a relational database, and has been widely studied [7, 15, 8, 2, 10]. Our approach is different from the existing approaches in that we consider semantic constraints in the XML model, and they are captured in our resulting relational schema. Further, our conversion is based on the strong and clean mathematical foundations provided by regular tree grammars.

In this paper, we first give a theoretical exposition for XML schemas using regular tree grammar theory, and provide two normal form representations – NF1 and NF2. The first step in several document validation algorithms based on tree automata [14] is to represent the given XML schema in NF1 to give better performance. Also the NF1 representation is used for checking the validity of a given schema against the constraints imposed by the schema language. For example, it is used to check whether a given tree grammar is a single type tree grammar [14].

An important application of NF2 presented in this paper is in the conversion from XML to relational models. However, the usefulness of NF2 is more general – NF2 provides the theoretical basis for mapping the type definitions in a language that supports union types, such as XML schemas, to a language that does not support union types, such as SQL.

After we describe the two normal form representations of regular tree grammars, we describe algorithms to map an XML schema, starting from the NF2 representation, to a relational schema. There are several issues in this mapping that we study:

– Given that the relational schema cannot express all the constraints in the XML schema, what is the useful and meaningful subset of constraints that should be mapped? Answering this question gives our schema simplification step, where the complex content model is converted into a simpler content model that can be represented in a relational schema.

- Inlining [15] is a technique that is used for generating more meaningful as well as "efficient" relational schemas. We describe the inlining algorithm which can be used for any general regular tree grammar.
- How do we handle collection types, recursion, and IDREF and IDREFS attributes defined in XML schema?
- How do we maintain semantic constraints such as key constraints and inclusion dependencies?

1.1 Related Work

Regular tree grammars and automata have been used by the authors in [14] to compare the expressive power and closure properties of the different XML schema language proposals. Furthermore, they are used for several document validation algorithms as presented in [14], and in several products such as the RELAX NG [5] validator, and XDuce [9] validator. The NF1 representation presented in this paper provides a useful addition to existing tools for document validation as well as schema validation.

Several techniques have been provided in the past for mapping an XML schema to a relational schema. In [7], the authors apply data mining techniques on semistructured data to find frequent patterns using a combination of the data instance as well as the query mix. The frequent patterns are stored in "optimized" relational storage, and the remaining patterns are stored in an overflow storage. This technique has the drawback that it requires integration of the relational and the overflow systems. In [8], the authors consider several mapping techniques, where an edge in the document $X\ e\ Y$ (i.e., Y is the value of the attribute of X called e) is mapped as three columns, X, e, Y. The drawback with this approach is the difficulty in maintaining semantic constraints. For example, suppose the value of the attribute e is a key for X, then it is difficult to specify this key constraint in the output relational schema.

The techniques presented in [15, 2] are closer to our goals of maintaining semantic constraints. We borrow some of our steps from them – our recursion elimination step is borrowed from [15], and the iterative improvement of the relational schema based on data and query statistics is borrowed from [2]. However, there are significant differences between our approaches. Maintaining semantic constraints was not the focus of both the above techniques, and hence they cannot capture several semantic constraints.

The techniques provided in [10] try to capture the semantic constraints in the relational schema. However they attempt to capture mainly cardinality constraints using equality generating dependencies, and tuple generating dependencies. Our work focuses on relationships and key constraints and maintains them in the relational schema.

1.2 Roadmap

In Section 2, we first define regular tree grammars. In Section 3, we define NF1 representation for a regular tree grammar, and describe how a regular tree gram-

mar can be converted to NF1. Also, we present two applications of the NF1 representation. In Section 4, we define NF2 representation for a regular tree grammar, and describe how we can obtain an NF2 representation for a given regular tree grammar. In Section 5, we describe the different steps in mapping a given XML schema to relational schema. Finally, concluding remarks and future directions are discussed in Section 6.

2 Regular Tree Grammar

The structural specification of an XML schema is a regular tree grammar. We define a regular tree grammar below, borrowed from [14].

Definition 1 (Regular Tree Grammar) *A regular tree grammar (RTG) is a 4-tuple $G = (N, T, S, P)$, where:*

- *N is a finite set of* non-terminals,
- *T is a finite set of* terminals,
- *S is a set of* start symbols, *where S is a subset of N,*
- *P is a finite set of production rules of the form $X \rightarrow a\ \ RE$, where $X \in N$, $a \in T$, and RE is a regular expression over N; X is the left-hand side, a RE is the right-hand side, and RE is the* **content model** *of this production rule.* □

Example 1. The grammar $G_1 = (N, T, S, P)$ given in Table 1 is a regular tree grammar. For instance, $Author1 \rightarrow author(Son^*)$ is a production rule, whose left-hand side, right-hand side and the content model are $Author1$, $author(Son^*)$, and (Son^*), respectively. □

A production rule $X \rightarrow a\ \ RE$ means that the non-terminal X can generate a tree with a as the root, and with children that "match" RE. In G_1, $Author1$ generates a tree with *author* as the root, and children that match Son^*. The set of trees that can be generated from any start symbol forms the language generated by the given regular tree grammar. For instance, a tree generated by G_1 is shown in Figure 1.

3 First Normal Form (NF1) for Regular Tree Grammars

In this section, we introduce the NF1 representation for regular tree grammars. NF1 representation requires that for every non-terminal, there must be *at most* one rule that produces a tree with a particular terminal as the root.

Definition 2 (NF1) *A regular tree grammar is said to be in* NF1 *if no two production rules have the same non-terminal in the left-hand side and the same terminal in the right hand side.* □

Table 1. An example regular tree grammar G_1

$$
\begin{array}{rl}
 & N = \{Book, Author1, Author2, Pub, \\
 & \quad Library, Museum, Son, Daughter\} \\
 & T = \{book, author, publisher, \\
 & \quad library, museum, son, daughter\} \\
 & S = \{Book\} \\
P: & Book \rightarrow book(Author1^*, Pub, \\
 & \quad (Library + Museum)) \\
 & Book \rightarrow book(Author2^*, Pub, Library) \\
 & Author1 \rightarrow author(Son^*) \\
 & Author2 \rightarrow author(Daughter^*) \\
 & Pub \rightarrow publisher(\epsilon) \\
 & Library \rightarrow library(\epsilon) \\
 & Museum \rightarrow museum(\epsilon) \\
 & Son \rightarrow son(\epsilon) \\
 & Daughter \rightarrow daughter(\epsilon)
\end{array}
$$

In other words, a regular tree grammar in NF1 does not have two rules of the form $X \rightarrow a\ \ RE_1$, and $X \rightarrow a\ \ RE_2$.

Example 2. The regular tree grammar G_1 in Example 1 is not in NF1. There are two rules, $Book \rightarrow book(Author1^*, Pub, (Library + Museum))$, and $Book \rightarrow book(Author2^*, Pub, Library)$, which have the same non-terminal in the left-hand side and the same terminal in the right hand side. □

Converting a given regular tree grammar to NF1 is straightforward: for every two rules of the form $X \rightarrow a\ \ RE_1$ and $X \rightarrow a\ \ RE_2$, replace the two rules by one rule as $X \rightarrow a\ \ (RE_1 + RE_2)$. This algorithm is outlined in Table 2.

Example 3. Converting G_1 to NF1, we obtain the regular tree grammar G_3 in Table 3.

The NF1 representation of regular tree grammars is used in at least two different XML application scenarios. One application scenario is in document

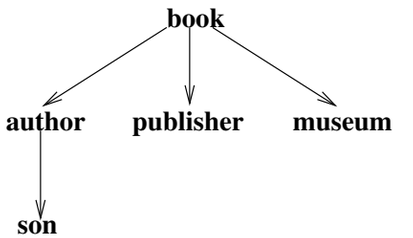


Fig. 1. An example tree that is generated by G_1

Table 2. RTG to NF1 algorithm

Convert a regular tree grammar $G = (N, T, S, P)$ to NF1. 1. If there does not exist two rules in G of the form $X \rightarrow a\ \ RE_1$, and $X \rightarrow a\ \ RE_2$, return G. 2. Else replace the two rules with one rule $X \rightarrow a\ \ (RE_1 + RE_2)$, and go to step 1.

Table 3. G_3 = NF1 representation for G_1

$$
\begin{array}{rrl}
 & N & = \{Book, Author1, Author2, Pub, \\
 & & \quad Library, Museum, Son, Daughter\} \\
 & T & = \{book, author, publisher, \\
 & & \quad library, museum, son, daughter\} \\
 & S & = \{Book\} \\
P: & Book & \rightarrow book((Author1^*, Pub, \\
 & & \quad (Library + Museum)) + \\
 & & \quad (Author2^*, Pub, Library)) \\
 & Author1 & \rightarrow author(Son^*) \\
 & Author2 & \rightarrow author(Daughter^*) \\
 & Pub & \rightarrow publisher(\epsilon) \\
 & Library & \rightarrow library(\epsilon) \\
 & Museum & \rightarrow museum(\epsilon) \\
 & Son & \rightarrow son(\epsilon) \\
 & Daughter & \rightarrow daughter(\epsilon)
\end{array}
$$

validation, where a given document is verified whether it is valid against a given XML schema. Different document validation algorithms are discussed in [14], which form the basis of several implementations. Many of these algorithms convert a given regular tree grammar into its NF1 representation as the first step for efficiency reasons. For example, consider the document validation algorithm based on non-deterministic bottom-up tree automata [14]. First, the given regular tree grammar is converted to NF1. During validation when a start tag is encountered, we identify those production rules $X \rightarrow a\ \ RE$, such that a is the terminal of this tag. There may be multiple production rules with a on the right hand side, but all these rules will have different non-terminals as the grammar is in NF1. When the corresponding end-tag is encountered, we check if the non-terminals assigned to the children belong to the language generated by RE. If so, X is one of the valid non-terminals for this element. Otherwise, X is not a valid non-terminal for this element. The advantage of the NF1 representation is that we need to check the validity of the non-terminals assigned to the children against fewer regular expressions, which would yield better performance. For example, the children of *book* have to be checked against two regular expressions in G_1, as opposed to one regular expression in G_3.

Another application of NF1 is in schema validation, where a given schema is verified whether it satisfies the constraints imposed by the schema language. Different schema language proposals impose different constraints on the possible set of XML schemas that are valid with respect to that schema language. For example, DTD forms local tree grammars, and XML-Schema forms single type tree grammars [14]. Now suppose given a schema, we have to check whether it is valid with respect to that language. For example, consider verifying whether G_1 is a valid schema in XML-Schema or not. So we have to check whether G_1 is a single type tree grammar. Single type tree grammars place the restriction that there *should* not exist two different non-terminals that are start symbols, or that occur in the content model of the same non-terminal that "compete" with each other[14]. Two non-terminals are said to compete with each other if they have production rules that generate trees with the same terminal as the root. To check if G_1 is a single type tree grammar, we first convert G_1 to its NF1 representation, G_3. Now we check whether there exists a production rule, where the non-terminals in its content model compete with each other. We find that non-terminals $Author1$ and $Author2$ compete with each other and occur in the same production rule. Therefore G_1 is not a valid schema if XML-Schema is used as the schema language.

4 Second Normal Form (NF2) for Regular Tree Grammars

In this section, we define the NF2 representation for regular tree grammars. NF2 forms the basis for conversion of type definitions in a programming language L_1 that supports union types (e.g., XML-Schema), to a programming language L_2 that does not support union types (e.g., SQL).

Definition 3 (NF2) *A regular tree grammar is said to be in* NF2 *if no production rule uses the union operator (denoted by "+") in its content model.* □

Example 4. The regular tree grammar G_1 is not in NF2. The content model $(Author1^*, Pub, (Library + Museum))$ uses the union operator and occurs in a production rule. □

Any given regular tree grammar can be converted to a regular tree grammar in NF2. The conversion algorithm for this is more involved than for NF1, and is given in Table 4. The algorithm uses a function migrateUnion(RE). migrateUnion(RE) is a recursive function which takes as input any regular expression RE, and returns an equivalent regular expression RE' of the form $(RE_1 + RE_2 + \ldots + RE_n)$, where no RE_i, $1 \leq i \leq n$ contains the union operator.

Example 5. Converting G_1 to NF2, we obtain the regular tree grammar G_5 given in Table 5. □

Mapping between XML and other models has become very important in recent years, with several applications exporting their data to XML, and several applications storing the XML data they obtain using other data models. NF2

Table 4. RTG to NF2 algorithm

Convert a regular tree grammar $G = (N, T, S, P)$ to NF2.

1. If there does not exist any rules in G of the form $R : X \rightarrow a \;\; RE$, where RE has the union operator, return G. Else go to Step 2.
2. Let migrateUnion$(RE) = (RE_1 + RE_2 \ldots RE_n)$.
3. Replace R with $\{X \rightarrow a \;\; RE_1, X \rightarrow a \;\; RE_2, \ldots, X \rightarrow a \;\; RE_n\}$, and go to Step 1.

migrateUnion : $RE \Longrightarrow RE'$, where $RE' = (RE_1 + RE_2 + \ldots + RE_n)$, and no RE_i, $1 \leq i \leq n$ contains the "+" operator

1. If RE does not contain "+", return RE.

2. If $RE = (r)^*$:
   Let migrateUnion$(r) = (r_1 + r_2 + \ldots + r_n)$.
   return $(r_1^*, r_2^*, \ldots, r_n^*)^*$.

3. If $RE = (r_1 + r_2)$:
   Let migrateUnion$(r_1) = (a_1 + a_2 + \ldots + a_m)$.
   Let migrateUnion$(r_2) = (b_1 + b_2 + \ldots + b_n)$.
   return $(a_1 + a_2 + \ldots + a_m + b_1 + b_2 + \ldots + b_n)$.

4. If $RE = (r_1, r_2)$:
   Let migrateUnion$(r_1) = (a_1 + a_2 + \ldots + a_m)$.
   Let migrateUnion$(r_2) = (b_1 + b_2 + \ldots + b_n)$.
   return $((a_1, b_1) + (a_1, b_2) + \ldots + (a_1, b_n) +$
   $(a_2, b_1) + (a_2, b_2) + \ldots + (a_2, b_n) + \ldots$
   $(a_m, b_1) + (a_m, b_2) + \ldots + (a_m, b_n))$.

provides the basis for mapping union types provided by XML schemas to types in the target model. It is useful when the target model does not support union types, such as the relational model. An overview of how this conversion works is as follows: the type definitions as provided by NF2 are mapped into type definitions in the target language. For example, G_1 that uses union types is converted using NF2 to G_5 which does not have union types. In G_5, there are three production rules with *Book* on the left-hand side. In the target language, there is a new type defined for each of these production rules, and thus there will be three types defined corresponding to *Book*, say $Book_1, Book_2, Book_3$. The language maintains additional book-keeping, this book-keeping assists in the mapping of operations from the original type definitions to the new ones. For example, a query such as $Book/Pub$ on the original XML schema will be mapped to, say, $(Book_1/Pub \cup Book_2/Pub \cup Book_3/Pub)$ in the new schema.

Table 5. G_5 = NF2 representation for G_1

$$
\begin{array}{rr}
 & N = \{Book, Author1, Author2, Pub, \\
 & \quad Library, Museum, Son, Daughter\} \\
 & T = \{book, author, publisher, \\
 & \quad library, museum, son, daughter\} \\
 & S = \{Book\} \\
P: & Book \rightarrow book(Author1^*, Pub, Library) \\
 & Book \rightarrow book(Author1^*, Pub, Museum) \\
. & Book \rightarrow book(Author2^*, Pub, Library) \\
 & Author1 \rightarrow author(Son^*) \\
 & Author2 \rightarrow author(Daughter^*) \\
 & Pub \rightarrow publisher(\epsilon) \\
 & Library \rightarrow library(\epsilon) \\
 & Museum \rightarrow museum(\epsilon) \\
 & Son \rightarrow son(\epsilon) \\
 & Daughter \rightarrow daughter(\epsilon)
\end{array}
$$

5 Mapping an XML Schema to Relational Schema

In this section, we describe our algorithm to map a given XML schema to a relational schema. Before we proceed, let us define *XSchema*, a language independent formalism to specify XML schemas. *XSchema* is based on regular tree grammar theory that we studied in the previous sections, and borrows from our definitions in [12, 13]. To define *XSchema*, we first assume the existence of a set $\widehat{E}$ of element names, a set $\widehat{A}$ of attribute names and a set $\widehat{\tau}$ of atomic data types defined in [1] (e.g., ID, IDREF, IDREFS, string, integer, date, etc). When needed, an attribute name $a \in \widehat{A}$ or an element name $e \in \widehat{E}$ is qualified by the element names using the *path expression* notation $e_1.e_2 \cdots e_n.a$, or $e_1.e_2 \cdots e_n.e$ where $e_i \in \widehat{E}, 1 \leq i \leq n$). *XSchema* extends regular tree grammars with the specification of data types, attribute definitions, primary key constraints, and inclusion dependency constraints. Further attributes of types IDREF and IDREFS identify the target types referred to by the values.

Definition 4 (*XSchema*) *An XSchema is denoted by 6-tuple* $\mathbb{X} = (E, A, M, P, r, \Sigma)$, *where:*

- E *is a finite set of element names in* $\widehat{E}$,
- A *is a function from an element name* $e \in E$ *to a set of attribute names* $a \in \widehat{A}$,
- M *is a function from an element name* $e \in E$ *to its* element type definition*: i.e.,* $M(e) = \alpha$, *where* α *is a regular expression:* $\alpha ::= \epsilon \mid \tau \mid \alpha + \alpha \mid \alpha, \alpha \mid \alpha^* \mid \alpha^? \mid \alpha^+$, *where* ϵ *denotes the empty element,* $\tau \in \widehat{\tau}$, *"+" for the union, "," for the concatenation, "*α^**" for the Kleene star,* $\alpha^?$ *for* $(\alpha + \epsilon)$ *and* α^+ *for* (α, α^*),
- P *is a function from an attribute name* a *to its* attribute type definition*: i.e.,* $P(a) = \beta$, *where* β *is a 4-tuple* (τ, n, d, f), *where* $\tau \in \widehat{\tau}$, n *is either "?"*

(nullable) or "¬?" (not nullable), d is a finite set of valid domain values of a or ϵ if not known, and f is a default value of a or ϵ if not known. Further more, if τ is IDREF *or* IDREFS*, then τ also specifies the target type or types that the attribute value should refer to using the symbol "$\rightsquigarrow$",*

- *$r \subseteq E$ is a finite set of root elements,*
- *Σ is a finite set of integrity constraints for XML model. The integrity constraints we consider are primary key constraints and inclusion dependencies.* □

Example 6. We shall use for this section the following *XSchema* and XML document. This schema represents a conference, and is slightly modified from the one in [10]. It serves as a good example for explaining the various steps in mapping an XML schema to a relational schema. The *XSchema* is given by $\mathbb{X}_6 = (E, A, M, P, r, \Sigma)$, where

$$
\begin{aligned}
E &= \{conf, title, date, editor,\\
&\quad\ paper, contact, author, person,\\
&\quad\ name, email, phone, cite\}\\
A(conf) &= \{id\}\\
M(conf) &= (title, date, editor^{?}, paper^{*})\\
P(conf.id) &= (\mathsf{ID}, \neg?, \epsilon, \epsilon)\\
M(title) &= (string)\\
A(date) &= \{year, mon, day\}\\
M(date) &= \epsilon\\
A(editor) &= \{eids\}\\
M(editor) &= (person^{*})\\
P(eids) &= (\mathsf{IDREFS} \rightsquigarrow (person^{*}), ?, \epsilon, \epsilon)\\
A(paper) &= \{id\}\\
M(paper) &= (title, contact^{?}, author, cite^{?})\\
P(paper.id) &= (\mathsf{ID}, \neg?, \epsilon, \epsilon)\\
A(contact) &= \{aid\}\\
M(contact) &= \epsilon\\
P(aid) &= (\mathsf{IDREF} \rightsquigarrow person, \neg?, \epsilon, \epsilon)\\
M(author) &= (person^{*})\\
A(person) &= \{id\}\\
M(person) &= (name, (email + phone)^{?})\\
P(person.id) &= (\mathsf{ID}, \neg?, \epsilon, \epsilon)\\
A(name) &= \{fn, ln\}\\
M(name) &= \epsilon\\
P(fn) &= (string, ?, \epsilon, \epsilon)\\
P(ln) &= (string, \neg?, \epsilon, \epsilon)
\end{aligned}
$$

$$
\begin{aligned}
M(email) &= (string)\\
M(phone) &= (string)\\
A(cite) &= \{id, format\}\\
M(cite) &= (paper^*)\\
P(cite.id) &= (\mathsf{ID}, \neg?, \epsilon, \epsilon)\\
P(format) &= (string, ?, (ACM|IEEE), \epsilon)\\
r &= \{conf, paper\}
\end{aligned}
$$

$$
\begin{aligned}
\Sigma = \{&title, date.year \overset{key}{\rightarrow} conf, title \overset{key}{\rightarrow} paper,\\
&name.ln \overset{key}{\rightarrow} person\}
\end{aligned}
$$

□

An XML document conforming to the above schema is:

```
<conf id="er05">
  <title>Int'l Conf on Conceptual Modeling</title>
  <date>
    <year>2005</year> <mon>Nov</mon> <day>25</day>
  </date>
  <editor eids="sheth bossy">
    <person id="klavans">
      <name fn="Judith" ln="Klavans"/>
      <email>klavans@cs.columbia.edu</email>
    </person>
  </editor>
  <paper id="p1">
    <title>Indexing Model for Structured ...</title>
    <contact aid="dao"/>
    <author>
      <person id="dao">
        <name fn="Tuong" ln="Dao"/>
      </person>
    </author>
  </paper>
  <paper id="p2">
    <title>Logical Information ...</title>
    <contact aid="shah"/>
    <author>
      <person id="shah">
        <name fn="Kshitij" ln="Shah"/>
      </person>
      <person id="sheth">
        <name fn="Amit" ln="Sheth"/>
        <email>amit@cs.uga.edu</email>
      </person>
    </author>
    <cite id="c100" format="ACM">
      <paper id="p3">
```

```
        <title Making Sense of Scientific ...</title>
        <author>
          <person id="bossy">
            <name fn="Marcia" ln="Bossy"</name>
            <phone>391.4337</phone>
          </person>
        </author>
      </paper>
    </cite>
  </paper>
</conf>
<paper id="p7">
  <title>Constraints Preserving ...</title>
  <contact aid="lee"/>
  <author>
    <person id="lee">
      <name fn="Dongwon" ln="Lee"/>
      <email>dongwon@cs.ucla.edu</email>
    </person>
  </author>
  <cite id="c200" format="IEEE"/>
</paper>
```

There are several steps in mapping an XML schema to a relational schema: (a) schema simplification, where we obtain from a given *XSchema*, a simpler *XSchema*, which will not have the constraints that cannot be captured in the relational model (b) inlining of elements and attributes, where we use simple "heuristics" to determine what are the attributes of a relation (c) mapping collection types, collection types can be represented in *XSchema* using * or $^+$. For example, in $M(A) = (\ldots, B^*, \ldots)$, A defines collection type of B. Collection types are considered as 1:many relationships and are mapped as such, (d) mapping IDREF and IDREFS attributes, IDREFS attributes are treated similar to child elements, (e) capturing the order specified in the XML model, and (f) enforcing constraints such as key constraints and inclusion dependencies.

Without loss of generality, we will assume for the rest of the section that the given *XSchema* is in NF2. We shall represent the element type definitions in the *XSchema* in NF2 as $(r_1 + r_2 + \ldots + r_n)$, rather than writing them out as multiple element type definitions. (This is equivalent to performing NF2, and then performing NF1 on the resulting schema.)

Example 7. When we represent $\mathbb{X}_6$ in NF2, the element type definitions change as: $M(conf) =$
$((title, data, paper^*) + (title, data, editor, paper^*))$
$M(paper) = ((title, author) + (title, contact, author) +$
$(title, author, cite) + (title, contact, author, cite))$
$M(person) = ((name) + (name, email) + (name, phone))$ □

Table 6. Schema Simplification Rules

Reg Exp	Simplified Reg Exp
$(r_1, r_2)[m, M]$	$(r_1[m, M], r_2[m, M])$
$(r_1[m_1, M_1])[m_2, M_2]$	$r_1[m_1 * m_2, M_1 * M_2]$
$(r_1[m_1, M_1], ..., r_1[m_2, M_2])$	$r_1[m_1 + m_2, M_1 + M_2]$

5.1 Schema Simplification

As mentioned before, the relational model cannot capture all the constraints specified in the *XSchema*. Our schema simplification step is based on the following principle: *For a parent-child relationship or* IDREFS *attribute, we capture only the cardinality of the child with respect to the parent as can be expressed using the two-tuple* $[minOccur, maxOccur]$. *We do not capture any other constraint that may be specified in the XSchema.*

For example, consider the element type definition $M(A) = (D, (B, C, B)^*)$. We do not capture: (a) the order constraint that a C must be followed and preceded by a B, (b) that the number of Bs must be twice the number of Cs, or (c) that the number of Bs should be even. We simplify the above content model as $M'(A) = (D, B^*, C^*)$. The simplification rules are applied to the target types identified by IDREFS attributes also.

Consider an element e, whose element type definition, after NF2 is given by $M(e) = (r_1 + r_2 + \ldots + r_n)$. We apply the schema simplification rules given below to every sub-expression r_i. Remember that NF2 ensures that r_i will contain only "," and "*" operators. We express the occurrence constraints using the notation $[minOccur, maxOccur]$ for convenience. This representation is equivalent to the occurrence constraints specifiable using regular expressions.
The *XSchema* after NF2 for $\mathbb{X}_6$ is already simplified, and the simplification rules described above do not change the schema.

5.2 Inlining

Inlining is used to generate more "meaningful" and efficient relational schemas. In inlining, we consider attributes of descendants of an element as attributes in the relation corresponding to that element. For example, consider the element *conf* which has child *date* which in turn has attributes *year*, *month*, and *day* in $\mathbb{X}_6$. Now we can inline the attributes of *date* to *conf* and obtain the relation $conf(year, month, date)$.

Inlining for an element e is done recursively using the function inline ($currEl$, $currSet$, $attSet$) described in Table 7. inline returns a set of relations that should be generated for an input element $currEl$. The function also takes as input $currSet$ which denotes the current set of relations we have, and $attSet$ which is used to maintain the list of attributes of e that should be present in every relation generated for e.

Table 7. Inline Function

inline : $currEl, currSet, attSet \Longrightarrow ResultSet$

1. Assign the set of attributes in $A(currEl)$ except IDREF and IDREFS attributes to $attSet$. Let the element type definition of $currEl$ be given by $M(currEl) = (r_1 + r_2 + \ldots + r_k)$. Set $ResultSet = \phi$.
2. For each r_i, we do the following.
   2.1. Set $currSet = attSet$.
   2.2. Let the elements which occur in r_i with occurrence constraint $[1, 1]$ after simplification be $\{e_1, e_2, \ldots, e_n\}$. For each e_i, do the following.
      2.2.1. If $M(e_i) \in \widehat{\tau}$, then $currSet = currSet \times e_i$.
      2.2.2. Else $currSet = currSet \times$ inline (e_i, ϕ, ϕ)
   2.3. If $currSet = \phi$, $currSet = currEl$.
   2.4. $ResultSet = ResultSet \cup currSet$.
3. return $ResultSet$.

To inline the element e, we call inline, where the initialization is: $currEl = e$, $currSet = \phi$, $attSet = \phi$.

A $ResultSet$ is returned by inline, and we define a relation corresponding to each term in the $ResultSet$. Note that when a subexpression r_i in $M(currEl)$ yields the empty set, we add the element name ($currEl$) as a placeholder. This ensures that inlining results in two relations for the element $conf$ in $\mathbb{X}_6$ as shown below.

Example 8. Suppose we perform inlining on $conf$, $paper$, and $person$, we obtain the following relation definitions. Note that inline for $M(editor)$ and $M(contact)$ actually produce the empty set, however inline will return $editor$ or $contact$ as placeholders.

$conf$: $conf1(id, title, year, mon, day)$,
$conf2(id, title, year, mon, day, editor)$.
$paper$: $paper1(id, title, author)$,
$paper2(id, title, contact, author)$,
$paper3(id, title, author, cite.id, format)$,
$paper4(id, title, contact, author, cite.id, format)$
$person$: $person1(id, fn, ln)$,
$person2(id, fn, ln, email)$,
$person3(id, fn, ln, phone)$ □

It is important to note that inlining could result in a huge number of relations. For instance, consider inlining an element a which has three choices, that is

$M(a) = (a_1 + a_2 + a_3)$. Let each of these choices in turn have three more choices and so on, that is, $M(a_1) = (a_{11} + a_{12} + a_{13})$. Let the height of this be n. The number of relations for this element will be 3^n. For example, if the height is 2, then we get nine relations. This could result in "inefficient" relational models. In a later subsection, we will study different cases when we place restrictions, so that we do not create so many relations. Also, our inlining is used for non-recursive elements. Recursive elements are handled in a later subsection.

5.3 Mapping Collection Types

Relational model cannot specify collection types. An element type definition such as $M(book) = (author^*)$ is captured in the relational model by defining separate relations for *book* and *author*, and a foreign key attribute for *author* referencing *book*. We use the same technique, however with some modifications. The modifications become necessary because we can have two different type definitions which specify collection type of the same sub-element. For example, consider $M(book) = (author^*)$, and $M(article) = (author^*)$. We will create two relations for *author*, one with a foreign key referencing *book* and another with a foreign key referencing *article*. We may even have an element type definition as $M(book) = (author^* + (author^*, publisher))$. When we do inlining for *book*, we end up with two relations, say $book1(book)$ and $book2(publisher)$. In this case again we create two relations for *author*, one referencing $book1$ and the other referencing $book2$. The algorithm for the conversion of collection types is as follows.

Let an element A have multiple occurrences in element type definitions, $m_1, m_2, \ldots, m_n$, where for each m_i, A has multiple occurrences in subexpressions for which the tables created are $m_{i1}, m_{i2}, \ldots, m_{in}$. Now a separate relation is created for A for each of these tables in each rule. Further, the table created for A corresponding to the table m_{ij} will define a foreign key referencing m_{ij}.

Example 9. In $\mathbb{X}_6$, we have $paper^*$ in the element type definition for $conf$ and *cite*. *paper* may occur as the root element also. Therefore, we have the following table definitions for *paper*: $paper1$, $paper2$, $paper3$ and $paper4$ represent papers that occur at the root of the document. In addition, we define papers that occur as children of conferences by defining all possible combinations of $\{paper1, paper2, paper3, paper4\}$ with $\{conf1, conf2\}$. For example, three of the tables that are defined are:

$paper1conf1(id, title, author, conf.title, year)$,
$paper1conf2(id, title, author, conf.title, year)$,
$paper2conf1(id, title, contact, author, conf.title, year)$
Here, *paper* and *cite* form a recursive relationship and this is handled in a different subsection. Just like for *paper* and $conf$, we define *person* which can occur as children of *editor* or author. □

5.4 Handling IDREF and IDREFS Attributes

Let us first consider IDREF attributes. An IDREF attribute specification will be of the form $B \rightarrow (@a :: \mathsf{IDREF} \rightsquigarrow (E_1 + E_2 + \ldots E_n))$, where $B \in E$, and E_i's either belong to E, or can be ϵ. Let the tables defined for any E_i be $m_{i1}, m_{i2}, \ldots, m_{in_i}$. Let the set of tables defined for B be $b_1, b_2, \ldots, b_n$. Our mapping replaces the set of tables for B with the set which is the cross product of $\{b_1, b_2, \ldots, b_n\}$ with the set of m_{ij}'s. Also for each of the resulting tables, we will have a foreign key referencing the ID attribute of m_{ij}.

Example 10. In $\mathbb{X}_6$, we have an IDREF attribute defined for *contact*, which refers to *person*. The set of tables defined for *contact* are $\{paper2, paper4\}$. Also we have three tables defined for *person* as $\{person1, person2, person3\}$. The result of our mapping is replacing *paper*2 and *paper*4 with the following six tables.

$$paper2person1(id, title, person1, author)$$
$$paper2person2(id, title, person2, author)$$
$$paper2person3(id, title, person3, author)$$
$$paper4person1(id, title, person1, author, cite.id, cite.format)$$
$$paper4person2(id, title, person2, author, cite.id, cite.format)$$
$$paper4person3(id, title, person3, author, cite.id, cite.format)$$

Here *person*1 column in each of the above tables is a foreign key referencing the *id* attribute of relation *person*1, *person*2 column references the *id* attribute of relation *person*2 and so on. □

IDREFS attributes are handled using the techniques for handling IDREF attributes and collection types, and techniques such as schema simplification and inlining. An IDREFS attribute specification is of the form $B \rightarrow (@a :: \mathsf{IDREFS} \rightsquigarrow RE)$, where $B \in E$, and RE is a regular expression over E. After schema simplification, let RE be given by $r = (r_1 + r_2 + \ldots + r_n)$. Now we perform a modified inlining, which we call IDREFSinline and is given in Table 8. IDREFSinline is a non-recursive function, and we call it as IDREFSinline (r).

Let the tables for B currently defined be $\{b_1, b_2, \ldots, b_n\}$. After we do IDREFSinline on an attribute of B, the set of tables for B change as follows. Suppose *ResultSet* returns $\{a_1, a_2, \ldots, a_n\}$. Let a_i be $(e_{i1}, e_{i2}, \ldots, e_{in_i})$. Let the set of tables for e_{ij} be denoted by the set T_{ij}. Now in *ResultSet*, we replace a_i with the set $T_{i1} \times T_{i2} \times \ldots \times T_{in_i}$. Now we replace the set of tables for B with the cross product of the sets $\{b_1, b_2, \ldots, b_n\}$ and *ResultSet*. Also in each of these tables, we define every column from *ResultSet*, say t_{ijk} as a foreign key referencing the ID attribute of the table t_{ijk}.

Now, consider an element say E that occurs as a collection type in r, we define a new set of tables, one for every possible combination of the tables of B and the tables of E. Each table has a set of columns representing the table in B,

Table 8. Inline Function for IDREFS attributes

IDREFSinline : $RE \Longrightarrow ResultSet$
1. Let $RE = (r_1 + r_2 + \ldots + r_n)$. For each r_i in $\{r_1, r_2, \ldots, r_n\}$, do the following. 1.1. Let the elements which occur in r_i with occurrence constraint $[1, 1]$ (after simplification) be $S = \{e_1, e_2, \ldots, e_n\}$. 1.2. If $S = \phi$, $ResultSet = ResultSet \cup currEl$. 1.3. Else $ResultSet = ResultSet \cup (e_1, e_2, \ldots, e_n)$. 2. return $ResultSet$.

say b_i, which will be a foreign key referencing the primary key of b_i, and a column representing the table of E, say e_i, which will be a foreign key referencing the ID attribute of e_i.

Example 11. In $\mathbb{X}_6$, we have an IDREFS attribute defined for *editor* as $@eids :: \mathsf{IDREFS} \rightsquigarrow person^*$. The IDREFS attribute has a collection type, so we define a set of new tables, one for every combination of $\{conf2\}$ and $\{person1, person2, person3\}$ as follows.

$eidsconf2person1(title, year, person1id)$
$eidsconf2person2(title, year, person2id)$
$eidsconf2person3(title, year, person3id)$ □

5.5 Handling Recursion

Recursion can occur in *XSchema* in two ways: through a cycle of elements with optional "?" occurrence, or through a cycle of elements with "*" occurrence. For example, $A \rightarrow (@a, A^?)$ forms a recursion of the first kind. We will handle this type of recursion using inlining. However simple inlining would result in infinitely many number of relations - we will create separate relations for recursion of depth 0, 1, 2, and so on. The relations will be $A1(@a)$, $A2(@a, @a)$, $A3(@a, @a, @a)$ and so on. Therefore, instead, we will use foreign keys - we will create relations of the form $A(@a, ARef)$, where $ARef$ refers to the A element that occurs as child of this element. For example consider the XML document fragment

```
<A a='a1'>
   <A a='a2'/>
</A>
```

When we translate this, we get the tuple, $(a1, a2ref)$ where $a2ref$ refers to $a2$. However, simply creating foreign keys also results in infinitely many number of relations. We will create separate relations for the A element which has no more children as $A1(@a)$, the A element that will have $A1$ as a child, say

$A2(@a, A1Ref)$, the A element that has $A2$ as a child, say $A3(@a, A2Ref)$, and so on. The above happens because we try to create many relations trying to enforce that we will not have null values in the relations. The solution is to create only one relation and allow null values for the columns. For example, the above relation will be translated to $A(@a, ARef)$ where $ARef$ is a foreign key referencing the relation A, and it can have null values.

The second kind of recursion is handled in a similar manner to how we handle collection types. For example, consider $A \rightarrow (@a, A^*)$. Our simple translation of collection types will again produce infinitely many relations – relations for As that do not have A as parents $A1(@a)$, As that have $A1$ as parent $A2(@a, A1Ref)$, As that have $A2$ as parent $A3(@a, A2Ref)$ and so on. This problem is solved again by enforcing one relation for A, as $A(@a, ARef)$, where $ARef$ is a foreign key referring to A and it can be null.

The general technique for handling recursion is similar to the one mentioned in [15]. For every strongly connected component, at least one of the elements must be defined as a separate relation. We enforce that such an element be mapped to exactly one relation. This is illustrated in Example 12. Further more, in a strongly connected component, if there exists an element which can be children of more than one element in the strongly connected component, then we define a separate relation for that element. This is necessary as we will see in Example 13.

Example 12. There is a cycle in $\mathbb{X}_6$ – formed by *paper*, and *cite*. This refers to the papers that are cited in a given paper. A separate relation is to be created for *paper*, which was already done. We enforce that there be only one relation corresponding to *paper*. Now we have to append all the attributes in the different tables corresponding to *paper* into one table. Doing this, we get one *paper* table as

$$paper(id, title, person1, person2, person3, cite.id, \\ format, conf1.title, conf1.year, conf2.title, \\ conf2.year, @paperRef).$$

Here $person1$, $person2$, $person3$ are nullable foreign keys, representing the contact author for a *paper*, $cite.id$, $format$ are nullable and represent the cite information, and $@paperRef$ is a nullable foreign key that refers to a paper that cited this paper. □

Example 13. Consider a simple 4 element schema, defined as $A \rightarrow (@a, B)$, $B \rightarrow (@b, A?, C?)$, $C \rightarrow (@c, D)$, $D \rightarrow (@d, A?, C?)$. This forms one strongly connected component with two nodes A and C having in-degree greater than one. Consider a sample document represented for convenience as

$$a1 \rightarrow b1 \rightarrow a2 \rightarrow b2 \rightarrow a3 \rightarrow b3 \rightarrow c1 \rightarrow d1 \rightarrow c2 \rightarrow d2 \rightarrow c3 \rightarrow d3 \rightarrow a4 \rightarrow b4 \rightarrow c4 \rightarrow d4$$

Here $x \rightarrow y$ means that y is a child of x. Also ai means an element A, with the value of the attribute $@a$ as ai.

Table 9. Relations from translation of Example 13. Here aR and cR are foreign keys referencing A and C respectively

A

@a	@b	aR	cR
$a1$	$b1$	$a2$	$null$
$a2$	$b2$	$a3$	$null$
$a3$	$b3$	$null$	$c1$
$a4$	$b4$	$null$	$c4$

C

@c	@d	aR	cR
$c1$	$d1$	$null$	$c2$
$c2$	$d2$	$null$	$c3$
$c3$	$d3$	$a4$	$null$
$c4$	$d4$	$null$	$null$

Suppose we define only one relation, as there is only one strongly connected component. Let us try to define the relation for A. We will see that we can have multiple C, D elements for one A, and this cannot be captured in the relation for A. This happens because there is a C, D loop, rather C has two parents, B and D. In this case, we define another relation for C. The two relations are given below.

5.6 Capturing Order Specified in the XML Model

It is necessary to maintain the order in which the elements occur in the document, so that we can reconstruct the document. Order can be captured in the XML model by keeping an order attribute corresponding to each element. The attribute which we maintain will give the position of the node in the whole document. However, capturing order is not the focus of this paper. So we will ignore it for the rest of this paper.

5.7 Capturing Semantic Constraints

XML schemas specify relationships using parent-child relationships, ID-IDREF/(S) attributes, and using inclusion dependencies. We have studied the mapping of all these features except inclusion dependencies. Inclusion dependencies are mapped as follows: Suppose the *XSchema* defines an inclusion dependency as: $A(X) \subseteq B(Y)$. Let the set of tables defined for B be $\{b_1, b_2, \ldots, b_n\}$, and the set of tables for A be $\{a_1, a_2, \ldots, a_n\}$. We replace the set of tables for A with the set formed by the cross product of $\{a_1, a_2, \ldots, a_n\}$ and $\{b_1, b_2, \ldots, b_n\}$. For any resulting table, say $a_i.b_j$, the set of columns is the same as the set of columns in a_i. Further, we define the inclusion dependency $a_i.b_j(X) \subseteq b_j(Y)$.

Key constraints are translated as follows. There are four kinds of tables that can be generated during our translation.

- *Table created corresponding to an element, the key for the element does not depend on any attribute of any of its ancestors*:
  Let the key for the element be denoted by K. K may consist of attributes, elements that are not collection types, elements that are collection types,

IDREF attributes, and IDREFS attributes. First, we remove the elements that are collection types and the collection type elements in IDREFS attributes. Then, we replace every element with its *key*. Let the resulting key be K'. Corresponding to every attribute in K', there will be a column defined for the table. The key for the table is defined as this set of columns corresponding to K'.

- *Table created corresponding to an element, the key for the element depends on at least one attribute of its ancestors.*
  This occurs for "relative keys" [3]. Relative keys can be explained with this example, consider a library schema with rules as $library \rightarrow (book^*)$, $book \rightarrow (@title, author^*)$, $author \rightarrow (@name)$. Let the key for *book* be $(@title)$. We could define the key for *author* as - for every *book*, the key for *author* is $(@name)$. Here, the key for *author* can be considered as $(@name, book)$. Now translation of the keys is just like in the previous step. Therefore in the relational schema, we get the key for *book* as $(@title)$, and the key for *author* as $(@name, bookRef)$.
- *Table created corresponding to collection type in* IDREFS *attributes*
  Consider $A \rightarrow (@a, @bRefs :: IDREFS \rightsquigarrow (B^*))$, $B \rightarrow (@b :: \mathsf{ID})$. For a collection type of an IDREFS attribute, a new table is created. Therefore we create a new table $bRefsAB(ARef, BRef)$, here $ARef$ is a foreign key referencing the primary key of A, and $BRef$ is a foreign key referencing the ID attribute of B. The key for this table will be the set of all columns of this table.
- *Table created corresponding to an element for which there is no key defined.*
  In this case, a system-generated identifier is introduced as the key for the table.

Example 14. In $\mathbb{X}_6$, for every table corresponding to $conf$, the key is $(title, year)$, for every table corresponding to *paper*, the key is $(title)$, and for every table corresponding to $person$, the key is (ln). Furthermore a set of tables are created corresponding to the IDREFS attribute *eids*, and the key for this is $(title, year, person)$. □

5.8 Decreasing the Number of Relations Generated

We have so far been generating multiple tables for an element. Also to handle recursion, only one table is created for an element in the recursion. Creating more tables decreases the number of null values in the resulting relation. However, often times the number of relations generated could be too many. So it is useful to restrict the number of relations produced for an element, though this will increase the number of null values. One extreme is enforcing that we create at most one table corresponding to an element. Or we can say that for every element that has a foreign key defined on it, we create only one table. We can also ask for user input, or use data and query statistics and iteratively improve the schema to give the "best" performance. Now, we will convert the example *XSchema*, $\mathbb{X}_6$

Table 10. Relations from translation of $\mathbb{X}_6$ and the example document

eids

title	year	ln
Int'l Conf. . .	2005	sheth
Int'l Conf. . .	2005	bossy

conf

id	title	year	mon	day
er05	Int'l Conf. . .	2005	Nov	25

paper

id	title	aid	cite.id	format	conf.title	year	paperRef
p1	Indexing Model. . .	dao	null	null	Int'l Conf. . .	2005	null
p2	Logical Information. . .	shah	c100	ACM	Int'l Conf. . .	2005	null
p3	Making Sense. . .	null	null	null	null	null	Logical Information. . .
p7	Constraints Preserving. . .	lee	c200	IEEE	null	null	null

person

id	fn	ln	email	phone	conf.title	year	paper.title
klavans	judith	klavans	klavans@cs. columbia.edu	null	Int'l Conf. . .	2005	null
dao	Tuong	Dao	null	null	null	null	Indexing Model. . .
shah	Kshitij	Shah	null	null	null	null	Logical Information. . .
sheth	Amit	Sheth	amit@cs. uga.edu	null	null	null	Logical Information. . .
bossy	Marcia	Bossy	null	391.4337	null	null	Making Sense. . .
lee	Dongwon	Lee	dongwon@cs. ucla.edu	null	null	null	Constraints Preserving. . .

and the document to the relational model. We will assume that for an element that has a foreign key defined on it, we will create at most one table.

5.9 Example

When we convert $\mathbb{X}_6$ and the document using our mapping algorithm, we obtain the following set of relations. We do not capture the order among elements in the document. Also for an element that has a foreign key defined on it, we will create at most one table. Therefore we will have only one table for *conf*, *paper* and *person*, as shown in Table 10. The constraints are:

$title, year \overset{key}{\rightarrow} conf$; $title \overset{key}{\rightarrow} paper$;
$ln \overset{key}{\rightarrow} person$; $title, year, ln \overset{key}{\rightarrow} eids$;
$eids.ln \subseteq person.ln$;
$eids.title, eids.year \subseteq conf.title, conf.year$;
$paper.aid \subseteq person.id$;

$paper.conf.title, paper.year \subseteq conf.title, conf.year$;
$paper.paperRef \subseteq paper.title$;
$person.conf.title, person.year \subseteq$
$conf.title, conf.year$;
$person.paper.title \subseteq paper.title$

5.10 Analysis of our Algorithm

Our first set of experiments were to determine the "goodness" of the relational schema we obtain. Our example illustrated that our conversion algorithm produces good relational schemas. We then took the TPC-H data and converted that to XML using the CoT algorithm [12]. Now we ran our XML to relational conversion on this XML data. We obtained the original relational data that we started off with.

We then analyzed the properties of our algorithm closely, and proved that when we convert the XML data resulting from CoT to relations using the techniques described in this paper, we will obtain the original relational schema that we started with. These results are quite promising for a common application scenario - person A wants to ship his data in a relational database to person B. A converts his data to XML and then ships this XML data. B receives this XML data, and converts it back to relations and stores it in his relational database. Now if A uses CoT for his relational to XML conversion, and if B uses the steps described in this paper for XML to relational conversion, then it is guaranteed that B will end up with the same relations as A started off with, no additional "handshake" between A and B is required.

6 Conclusion

In this paper, we presented a theoretical basis for different XML applications using regular tree grammars. We defined the NF1 representation, which is useful in document validation as well as schema validation. Our NF2 representation forms the basis for mapping type definitions in XML schemas to a language that does not provide union types, such as SQL. Further, we described the steps in mapping an XML schema to a relational schema, while maintaining semantic constraints. Prelimiary studies indicate that our algorithm generates "good" relational schema, and further it complements our relational to XML conversion algorithm, CoT.

There are still several issues to be studied with respect to data modeling using XML schemas. One interesting question is what restrictions can be imposed on the XML schemas for data modeling. For example, ability to specify recursive types is one of the advantages of the XML data model, but CoT generates XML schemas that have no recursion. Also there are several important questions regarding operations for the XML model. What is a set of "good" operations for the XML model? How do we map operations on the XML model to the relational model? Answering these questions will clarify the data modeling aspects of XML schemas.

References

[1] P. V. Biron and A. Malhotra (Eds). "XML Schema Part 2: Datatypes". W3C Recommendation, May 2001. http://www.w3.org/TR/xmlschema-2/. 89

[2] P. Bohannon, J. Freire, P. Roy, and J. Simeon. "From XML Schema to Relations: A Cost-Based Approach to XML Storage". In *IEEE ICDE*, San Jose, CA, Feb. 2002. 82, 83

[3] P. Buneman, S. Davidson, W. Fan, C. Hara, and W. Tan. "Keys for XML". In *Int'l World Wide Web Conf. (WWW)*, Hong Kong, May 2001. 100

[4] D. Chamberlin, D. Florescu, J. Robie, J. Siméon, and M. Stefanescu (Eds). "XQuery 1.0: An XML Query Language". W3C Working Draft, Jun. 2001. http://www.w3.org/TR/2001/WD-xquery-20010607/. 81

[5] J. Clark and M. Murata (Eds). "RELAX NG Tutorial". OASIS Working Draft, Jun. 2001. http://www.oasis-open.org/committees/relax-ng/tutorial.html. 81, 83

[6] H. Comon, M. Dauchet, R. Gilleron, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. "Tree Automata Techniques and Applications", 1997. http://www.grappa.univ-lille3.fr/tata. 81

[7] A. Deutsch, M. F. Fernandez, and D. Suciu. "Storing Semistructured Data with STORED". In *ACM SIGMOD*, Philadephia, PA, Jun. 1998. 82, 83

[8] D. Florescu and D. Kossmann. "Storing and Querying XML Data Using an RDBMS". *IEEE Data Eng. Bulletin*, 22(3):27–34, Sep. 1999. 82, 83

[9] H. Hosoya and B. C. Pierce. "XDuce: A Typed XML Processing Language". In *Int'l Workshop on the Web and Databases (WebDB)*, Dallas, TX, May 2000. 81, 83

[10] D. Lee and W. W. Chu. "CPI: Constraints-Preserving Inlining Algorithm for Mapping XML DTD to Relational Schema". *J. Data & Knowledge Engineering (DKE)*, 39(1):3–25, Oct. 2001. 82, 83, 90

[11] D. Lee, M. Mani, F. Chiu, and W. W. Chu. "Nesting-based Relational-to-XML Schema Translation". In *Int'l Workshop on the Web and Databases (WebDB)*, Santa Barbara, CA, May 2001. 82

[12] D. Lee, M. Mani, F. Chiu, and W. W. Chu. "NeT & CoT: Translating Relational Schemas to XML Schemas using Semantic Constraints". Technical report, UCLA Computer Science Dept., Feb. 2002. 82, 89, 102

[13] M. Mani, D. Lee, and R. D. Muntz. "Semantic Data Modeling using XML Schemas". In *Int'l Conf. on Conceptual Modeling (ER)*, Yokohama, Japan, Nov. 2001. 81, 89

[14] M. Murata, D. Lee, and M. Mani. "Taxonomy of XML Schema Languages using Formal Language Theory". In *Extreme Markup Languages*, Montreal, Canada, Aug. 2001. http://www.cs.ucla.edu/∼dongwon/paper/. 81, 82, 83, 84, 86, 87

[15] J. Shanmugasundaram, K. Tufte, G. He, C. Zhang, D. DeWitt, and J. Naughton. "Relational Databases for Querying XML Documents: Limitations and Opportunities". In *VLDB*, Edinburgh, Scotland, Sep. 1999. 82, 83, 98

[16] M. Takahashi. "Generalizations of Regular Sets and Their Applicatin to a Study of Context-Free Languages". *Information and Control*, 27(1):1–36, Jan. 1975. 81

[17] H. S. Thompson, D. Beech, M. Maloney, and N. Mendelsohn (Eds). "XML Schema Part 1: Structures". W3C Recommendation, May 2001. http://www.w3.org/TR/xmlschema-1/. 81

Adaptive XML Shredding: Architecture, Implementation, and Challenges

Juliana Freire* and Jérôme Siméon

Bell Laboratories
600 Mountain Avenue, Murray Hill, NJ 07974, USA
{juliana,simeon}@research.bell-labs.com

1 Introduction

As XML data becomes central to business-critical applications, there is a growing need for efficient and reliable XML storage. Two main approaches have been proposed for storing XML data: native and colonial systems. *Native systems* (*e.g.,* [9, 20]) are designed from the ground up specifically for XML and XML query languages. *Colonial systems* (*e.g.,* [5, 7, 19]), on the other hand, attempt to reuse existing commercial database systems (DBMS) by mapping XML into the underlying model used by the DBMS. Colonial systems can thus leverage features, such as concurrency control, crash recovery, scalability, and highly optimized query processors available in the DMBS, making them an attractive alternative for managing XML data. However, several technical challenges need to be addressed in terms of architecture, algorithms, and implementation of these systems. In this paper, we described how these issues are addressed in the context of colonial systems that use relational databases as the underlying DBMS.

The mismatch between the XML and the relational models implies that one must first *shred* an XML tree-structured document so that it *fits* into flat relational tables. Therefore, a mechanism is needed to determine the appropriate storage configuration. Once a mapping is selected, the system must provide support for loading the XML data into the database, and to translate queries over the original document into queries over the mapped data. There are different approaches for these problems. For example, while commercial relational systems require users to manually define mappings [14, 15], techniques have been proposed to automatically derive XML-to-relational mappings that adopt either a fixed shredding strategy [19, 11] or that derive the best shredding for a given application [5, 4]. Different techniques have also been proposed for query translation [10, 6].

Although individual problems pertaining to colonial XML storage systems have been studied in isolation, to the best of our knowledge, the design and implementation of a complete colonial system has not been described in the literature. In this paper, we discuss the design and implementation of LegoDB [5], a colonial XML data management system. In particular, we present the complete

* Current address: juliana@cse.ogi.edu.

S. Bressan et al. (Eds.): EEXTT and DIWeb 2002, LNCS 2590, pp. 104–116, 2003.

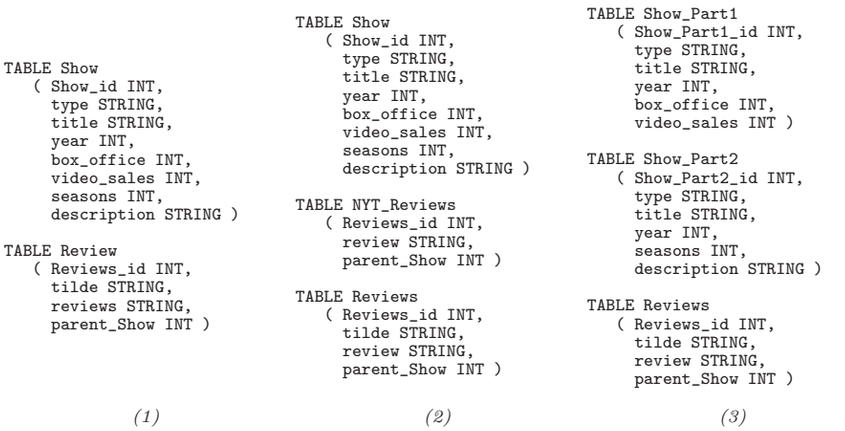


Fig. 1. Three storage configurations for the movie database

architecture of the system, its implementation, and the describe underlying algorithms.

The paper is organized as follows. Section 2 motivates the need for adaptive XML shredding. Section 3 presents the general architecture of a colonial system that supports adaptive shredding. Sections 4, 5 and 6 describe the components of such an architecture and their implementation in the LegoDB system.

2 The Need for Adaptive Shredding

In this section, we motivate the need for adaptive shredding through a scenario inspired from the Internet Movie Database (`http://www.imdb.com`), which provides access to information about movies and television shows. A fragment of Document Type Definition (DTD) corresponding to this document is illustrated below.

```
<!ELEMENT imdb (show*, director*, actor*)>
<!ELEMENT show (title, year, reviews*,
                ((box_office, video_sales)
                |(seasons, description, episode*)))>
<!ATTLIST show type CDATA #REQUIRED>

<!ELEMENT title (#PCDATA)>
<!ELEMENT year (#PCDATA)>
<!ELEMENT review (#PCDATA)>
....
```

An IMDB document contains a collection of shows, movie directors and actors. Each show can be either a movie or a TV show. Movies and TV shows have some information in common (*e.g.,* `title` and `year` of production), but they also contain information that is specific to each (*e.g.,* movies have a `box_office` and `video_sales`, TV shows have `seasons`).

There are many different ways to store data that conforms with the IMDB DTD in a relational database. Figure 1 shows three alternatives. The first configuration results from inlining as many elements as possible in a given table,

roughly corresponding to the shared strategy proposed in [19]. The second configuration is obtained from the first by partitioning the `Reviews` table into two tables (one that contains New York Times reviews, and another for reviews from other sources). Finally, the third configuration is obtained from the first by splitting the `Show` table into `Show_Part1` for movies, and `Show_Part2` for TV shows.

Even though a given configuration can be efficient for one application, it may lead to poor performance for others. Thus, it is not possible to select the *best* configuration in isolation, without taking the application characteristics and *cost* into account. As an illustration, consider the following XQuery [3] queries:

```
Q1:
for  $v in imdb/show
where $v/year = 1999
return ($v/title, $v/year,
        $v/nyt_reviews)

Q3:
for $v in imdb/show
where $v/title = c3
return $v/description
```

```
Q2:
for $v in imdb/show return $v

Q4:
for $v in imdb/show
return <result>
        { $v/title, $v/year,
          (for $e in $v/episode
             where $e/guest_director = c4
           return $e) }
       </result>
```

Queries Q1 and Q2 are typical of a publishing scenario as in [10] (*i.e.,* to send a movie catalog to an interested partner). Queries Q3 and Q4 contain selection criteria and are typical of interactive lookup queries, such as the ones issued against the IMDB Web site itself. We then define two workloads, $\mathsf{W1} = \{Q1 : 0.4, Q2 : 0.4, Q3 : 0.1, Q4 : 0.1\}$ and $\mathsf{W2} = \{Q1 : 0.1, Q2 : 0.1, Q3 : 0.4, Q4 : 0.4\}$, where each workload contains a set of queries and an associated weight that reflects the importance of each query for the application.

Figure 2 shows the estimated costs for each query and workload, as returned by the LegoDB optimizer for each configuration in Figure 1 (costs are normalized by the costs of mapping 1). Only the first of the three storage mappings shown in Figure 1 can be generated by previous heuristic approaches. However, this mapping has significant disadvantages for either workload we consider. First, due to its treatment of union, it inlines several fields which are not present in all the data, making the `Show` relation wider than necessary. Second, when the entire `Show` relation is exported as a single document, the records corresponding to movies need not be joined with the `Episode` tables, but this join is required by the first two configurations. Finally, the large `Description` element need not be

	Configuration 1	Configuration 2	Configuration 3
Q1	1.00	0.83	1.27
Q2	1.00	0.50	0.48
Q3	1.00	1.00	0.17
Q4	1.00	1.19	0.40
W1	1.00	0.75	0.75
W2	1.00	1.01	0.40

Fig. 2. Performance of Different Configurations

inlined unless it is frequently queried. The principle behind adaptive shredding is to automatically explore a space of possible relational configurations, and select the configuration that leads to the lowest cost for evaluating the application workload.

3 Principles and Architecture

A colonial XML storage system has two main components: storage design and run-time. The main task of the design component is to generate a target relational schema to store the input XML document. In adaptive shredding systems such as LegoDB, this component is rather complex. It takes into account information about the target application to both generate a space of possible mappings and evaluate the effectiveness of the derived mappings.

Figure 3(a) shows the architecture of the design component in LegoDB. The LegoDB storage engine takes as inputs: an XML Schema, an XQuery workload, and a set of sample documents. As output, it produces an *efficient* relational configuration (a set of relational tables) as well as a mapping specification. The modules of the storage design components are described below.

StatiX The first task in the system is to extract *statistical information* about both the values and structure of the input XML document. This information is used to derive *precise* relational statistics that are needed by the relational optimizer to accurately estimate the cost of the query workload. In LegoDB, this is done by the *StatiX* module, described in Section 5.

Schema Normalization The statistics together with the XML Schema are sent to the *Schema Normalization* module, which produces a *physical schema* (p-schema). P-schemas extend XML Schemas with statistical information, and

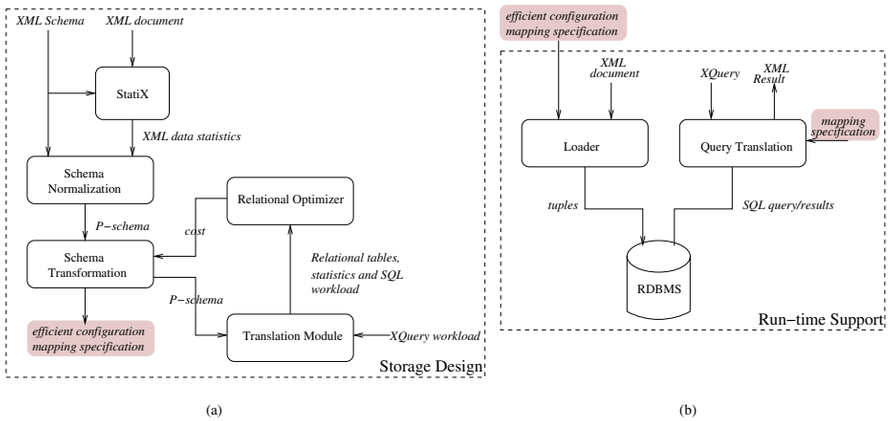


Fig. 3. Design and Run-Time Architectures

have a structure such that each type defined in the schema can be directly mapped into a relational table (see Section 4).
Schema Transformation The system searches for an efficient relational configuration by repeatedly transforming p-schemas, *i.e.,* generating new p-schemas that are structurally different, but that validate the same documents. Each new p-schema corresponds to a possible relational configuration. XML Schema transformations and search algorithms are discussed in Section 6.
Translation Module For each p-schema, the *Translation Module* generates a corresponding relational schema, translates the XQuery workload into SQL, and derives the appropriate relational statistics.
Relational Optimizer LegoDB uses a standard relational optimizer for cost estimation as a black box. As a result, it is possible to use LegoDB with different databases whose optimizers have different cost-models. The quality of the derived configurations depends on the accuracy of the estimates computed by the optimizer.

The design module produces a specification for the mapping that has the lowest cost among the alternatives explored by LegoDB. This specification is then used by the Loader and Query Translation modules (see Figure 3(b)) to create and populate the corresponding tables in the RDBMS, and answer application queries.

Loader Given a mapping specification and an XML document, the Loader module populates the target relational database. As described in Section 4.3, the Loader extends an XML Schema validating parser to generate tuples a document is validated.
Query Translator The *Query Translation* module, described in Section 4.2 is used to perform query translation on behalf of the target XML application.
Note that other tools for mapping XQuery to SQL (*e.g.,* [10]) could be used in LegoDB.

4 XML-to-Relational Translation

4.1 Mapping Schemas

LegoDB [5] generates a space of possible schema mappings by repeatedly transforming the original XML Schema, and for each transformed schema, applying a *fixed mapping* from XML Schema into relations. In this section, we describe how the fixed mapping is implemented. Details of the schema transformations and search algorithm are given in Section 6.

In order to guarantee the existence of a fixed mapping, we define the notion of *physical schema* (*p-schema*). In a p-schema, each type name defines a structure that can be directly mapped to a relation. More precisely, this structure must be such that it contains only (possibly nested) singleton elements with a simple value; (possibly nested) optional elements with a simple value; and all complex regular expressions may only contain type names. This last condition ensures

that complex regular expressions, such as union and repetition, do not contain actual values. As a result, a schema that verifies such criteria can be mapped to relations by creating a table for each type, and creating a column in that table for each element with a simple value. In the examples that follow, for simplicity, we use the type notation from the XQuery type system, described in [8]. A formal definition of p-schemas and the corresponding grammar appear in [5].

We now sketch the *normalization* algorithm used to generate a *p-schema* from an XML Schema. The algorithm is applied top-down on the structure of the type, and for each type in the original schema. For a type definition `define type X { T }`, where `X` is the name of the type, and `T` is the structure defining that type, the normalization algorithm is first applied on `T`. This returns a new type `T'` along with a set of new type definitions `define type X1 { T1 }` ... `define type Xn { Tn }`, where all of the type `T'`, `T1`, ..., `Tn` are normalized. A given type structure `T` is normalized, recursively, as follows:

– If the type is an atomic type (*e.g.,* `string`), return the type unchanged.
– If the type is an (optional) element declaration, then return the element declaration with its content normalized.
– If the type is a sequence of two types, return the sequence of their normalized types.
– If the type is a repetition (*e.g.,* `element a*`), then insert a new type name with the normalized content of the original type (*e.g.,* `X1*` with `define type X1 { element a }`).
– If the type is a union (*e.g.,* `element a | element b`), then create a new type name for each component of the union with the contents of the original type normalized (*e.g.,* `X1 | X2` with `define type X1 { element a }` and `define type X2 { element b }`).

After the original schema is normalized, transformations can be applied on the resulting p-schema. These transformations always result in a p-schema which can then be mapped into a relational configuration. The algorithm to map a p-schema into a set of relations is as follows:

– Create one relation $R_{\texttt{X}}$ for each type name `X`;
– For each relation $R_{\texttt{X}}$, create a key that stores the id of the corresponding element;
– For each relation $R_{\texttt{X}}$, create a foreign key parent_$P_{\texttt{X}}$ to all the relations $R_{P_{\texttt{X}}}$ such that $P_{\texttt{X}}$ is a parent type of `X`;
– Create a column in $R_{\texttt{a}}$ for each element `a` inside the type `X` that contains a value;
– If the data type is contained within an optional type then the corresponding column can contain a null value.

4.2 Mapping Queries

The LegoDB prototype implements a simple version of query translation for a fragment of XQuery which consists of: simple path navigation, selections, joins,

and nested queries. More sophisticated query mapping techniques such as the ones proposed in [10, 6] can be integrated into the system.

Translating an XQuery query into SQL requires analysis of the XPath expressions contained in the query and information about the schema mapping in order to determine the relational tables and columns to be accessed. In LegoDB, the mapping of XQuery to SQL is done in two phases. The first phase rewrites an XQuery XQ in a normal form XQ_{nf}. For example, the query:

```
for $show in document("www.imdb.com/imdb.xml")/show
where $show/year >= 2000
return $show/title, $show/reviews, $show/box_office
```

is normalized into:

```
let $imdbdoc := document("www.imdb.com/imdb.xml")
for $show in $imdbdoc/show, $v_title in $show/title,
    $v_box in $show/box_office, $v_year in $show/year,
    $v_reviews in $show/reviews
where $v_year >= 2000
return ($v_title, $v_reviews, $v_box)
```

XQ_{nf} is then translated into an equivalent SQL query for the given p-schema:

– *SELECT clause.* For each variable v in the XQuery `return` clause, if v refers to a type in the *p-schema*, all attributes of the corresponding table are added to the clause. Otherwise, if v refers to an element with no associated type, the corresponding attribute is added to the clause.
– *FROM clause.* For each variable v mentioned in the XQuery (in both `where` and `return` clauses), if v refers to a type T in the *p-schema*, the corresponding table R_T is added to the clause.
– *WHERE clause.* Conditions in `where` clause are translated in a straightforward manner, by replacing variable occurrences with the appropriate column name. In addition, for each variable in the XQuery, if the path expression defining the variable includes elements in that are mapped into separate tables, a condition must be added to enforce the key/foreign-key constraint.

For example, given the third configuration in Figure 1, the query mapping algorithm generates the following SQL Query:

```
SELECT S1.title, S1.box_office, R.Reviews_id
FROM Show_Part1 S1, Reviews R
WHERE S1.Show_Part1_id = R.parent_Show AND S1.year >= 2001
```

4.3 Mapping Data

After a mapping is selected by LegoDB, the next step is to create the database tables and load the XML documents. LegoDB extends an XML Schema validating parser to generate tuples as documents are validated.[1] Validation is a basic

[1] In LegoDB, both data loading and statistics gathering are performed during document validation (see Section5).

operation for XML documents. The most important property we use from XML Schema validation is the notion of type assignment: during validation each node in the XML tree is assigned a unique type name from the schema. Since the mappings are type-based, and one relation is created per type, the parser outputs a tuple for each instance of type encountered during validation. We describe below the process used in the current prototype to generate insert statements that populate the target database.

– Validation is performed top down starting at the root of the document.
– For each type name in the schema, a structure that represents a tuple is kept in memory to store the current part of the type which is validated. This structure is created based on the fixed mapping described in Section 4.
– Each time validation reaches a value that corresponds to an attribute of a tuple, the tuple is populated accordingly.
– When the tuple is full, it is flushed on disk as an insert statement.
– This process is repeated until the document is fully validated.

Note that currently this phase is done in LegoDB using a batch file. It is also possible to directly populate the database using ODBC, or to use more efficient bulk-loading facilities.

5 Statistics and Cost Estimation

LegoDB uses a standard relational optimizer [18] for cost estimation. But in order to derive accurate cost estimates, the optimizer needs accurate statistics. In LegoDB, the StatiX module is responsible for gathering statistics about the input XML document. As described in [12], it is important that these statistics capture both value and structural skews present in the data, and that they contain enough information so that accurate relational statistics can be derived as transformations are applied to the XML Schema.

Similar to data loading, statistics gathering is performed simultaneously with document validation. StatiX gathers statistics on a *per-type* basis. As a document is validated, globally unique identifiers (IDs) are assigned to all instances of the types defined in the schema; and together with these ID assignments, the system keeps track of the cardinality of each edge in the XML Schema type graph. Using this information, *structural histograms* constructed which use the IDs to summarize information about how elements are connected. Note that, while StatiX uses histograms in a novel way to summarize structural information, histograms are also used in a more traditional sense: *value histograms* can be built for types that are defined in terms of base types (*e.g.,* `Integer`).

Statistics gathering proceeds as follow. First, each distinct type in the XML Schema is assigned a unique type ID off-line. Then, for each type ID, we maintain a structure which contains: (1) a counter for the next available ID for that type, and (2) a set of all parent IDs that have been processed for that type. During document validation, the validation function is given the current parent ID. Whenever a new instance of the type is encountered, it is assigned a local ID

using the sequential counter associated with the type, and the current parent ID is added to the parent set for the corresponding type. The global ID of the element is then simply obtained by concatenating the associated type id and its local id. The modified part of the standard validation procedure is given in pseudo-code below.

```
   /* Type struct is a~counter plus parent types */
type TypeStructure = { counter : integer; parents : [ID] }

   /* We maintain a~type structure for each type */
AllTypes := [ (typename,type_structure) ]

   /* Maintains the type structure and returns the new type ID */
fun add~to~type(T : Type, ParentID : ID)
{ TS := find~type~structure(AllTypes,T);  /* get the type structure for T */
   NewID := build~id(T, TS.counter);        /* creates new ID */
   TS.counter := TS.counter+1;   /* increments the counter */
   TS.parents := add(ParentID, TS.parents)   /* adds parent ID */
   return NewID;  }

   /* Here is a~part of the validation function */
fun validate~sequence (S : Sequence, R : RegExp, ParentID : ID)
{ Deriv := derivatives(R); /* computes the derivatives */

   Node  := first(S);  /* get the first node */
   (NodeType,R') := match(Deriv,Node);  /* find the node type for the node */

   CurrentID := add~to~type(NodeType, ParentID);  /* get the ID of the node */

   validate~node(Node,NodeType,CurrentID);  /* Validate the children of the node */
   validate~sequence(rest(S), R', ParentID);      /* validates the rest */ }
```

The structural information gathered during validation can be summarized using standard histogram techniques. A variety of histogram constructions have been described in the literature [17] – the most common are *equi-width* histograms, wherein the domain range covered by each histogram bucket is the same, and *equi-depth* histograms, wherein the frequency assigned to each bucket is the same. Since it has been shown in [16] that equi-depth histograms result in significantly less estimation error as compared to equi-width histograms, we have implemented the former in StatiX.

6 Searching

In this section, we describe how a space of alternative shreddings is constructed and techniques to search for the best shredding for a given application.

6.1 Transforming P-schemas

There are many different schemas that validate a given document. For instance, different but equivalent regular expressions (*e.g.,* `(a(b|c*)) ((a,b)|(a,c*))`) can describe the contents of a given element. In addition, the presence or absence of a type name does not change the semantics of the XML Schema.

By transforming regular expressions that define XML elements, and by adding and removing types, LegoDB derives a series of equivalent schemas. These schemas

have *distinct type structures* that when mapped into relations using the fixed type-to-relation mapping (Section 4), lead to distinct relational configurations. There is large number of possible transformations that can be applied to XML Schemas, and we describe some below. For a more comprehensive discussion on transformations used in LegoDB, the reader is referred to [5].

Inlining/Outlining. In an XML Schema, associating a type name to a given nested element (outlining) or nesting its definition directly within its parent element (inlining) does not change the semantics of the schema, *i.e.,* the modified schema validates the same set of documents. However, rewriting an XML Schema in that way impacts the mapped relational schema by inlining or outlining the corresponding element within its parent table. Inlining is illustrated below:

```
define type TV {
 element seasons { int },
 type Description,
 type Episode*
}
define type Description {
 element description { string }
}
```

→

```
define type TV {
 element seasons { int },
 element description { string },
 type Episode*
}
```

At the relational level, this rewriting corresponds to the following transformation:

```
TABLE TV
  ( TV_id INT,
    seasons STRING,
    parent_Show INT)

TABLE Description
  ( Description_id INT,
    description STRING,
    parent_TV INT)
```

→

```
TABLE TV
  ( TV_id INT,
    seasons STRING,
    description STRING,
    parent_Show INT)
```

Note that inlining is the basis for the strategies proposed in [19]. It reduces the need for joins when accessing the content of an element, but at the same time it increases the size of the corresponding table and the cost of retrieving individual tuples. In the example above, the benefits of inlining or outlining `description` element within the `TV` type depend both on the access frequency for this element in the workload as well as its length.

Union Factorization/Distribution. Union types are often used to add some degree of flexibility to the schema. As queries can have different access patterns on unions, *e.g.,* access either parts together or independently, it is essential that appropriate storage structures are derived. LegoDB uses simple distribution laws on regular expressions to explore alternative storage structures for union. The first law (`(a,(b|c)) == (a,b | a,c)`) allows distribution of a union within a regular expression. The second law (`a[t1|t2] == a[t1]|a[t2]`) allows to distribute a union over an element (crossing element boundaries). For example, in Figure 1, applying these two laws on configuration 1 leads to configuration 3. The union transformations highlight the advantages of working directly at the XML Schema level. The horizontal partitioning of the relational schema derived from the configuration (3) in Figure 1 would not be easily found by a relational physical-design tool, since the information about the set of attributes involved in the union would not be available in the relational setting.

6.2 Searching for the Best Shredding

By repeatedly applying schema transformations, LegoDB generates a space of alternative *p-schemas* and corresponding relational configurations. Since this space can be very large (possibly infinite), it is not feasible to perform an exhaustive search. As shown in Algorithm 6.1, LegoDB uses a greedy heuristic to prune the search space. The algorithm begins by deriving an initial configuration $pSchema$ from the given XML Schema $xSchema$ (line 3). Next, the cost of this configuration, with respect to the given query workload $xWkld$ and the data statistics $xStats$ is computed using the function $GetPSchemaCost$ which will be described in a moment (line 3). The greedy search (lines 5-16) iteratively updates $pSchema$ to the cheapest configuration that can be derived from $pSchema$ using a single transformation. Specifically, in each iteration, a list of candidate configurations $pSchemaList$ is created by applying all applicable transformations to the current configuration $pSchema$ (line 7). Each of these candidate configurations is evaluated using $GetPSchemaCost$ and the configuration with the smallest cost is selected (lines 8-14). This process is repeated until the current configuration can no longer be improved. Note the use of the function $ApplyTransformations$, which applies the schema transformations described above.

Algorithm 6.1 Greedy Heuristic for Finding an Efficient Configuration

```
   Procedure GreedySearch
   Input:    xSchema : XML Schema,
             xWkld : XML query workload,
             xStats : XML data statistics
   Output:   pSchema : an efficient physical schema
1  begin
     minCost = ∞;
     pSchema = GetInitialPhysicalSchema(xSchema)
     cost = GetPSchemaCost(pSchema, xWkld, xStats)
5    while (cost < minCost) do
       minCost = cost
       pSchemaList = ApplyTransformations(pSchema)
       for each pSchema' ∈ pSchemaList do
         cost' = GetPSchemaCost(pSchema', xWkld, xStats)
10       if cost' < cost then
           cost = cost'
           pSchema = pSchema'
         endif
       endfor
15   endwhile
     return pSchema
   end.
```

$GetPSchemaCost$ computes the cost of a configuration given a $pSchema$, the XML Query workload $xWkld$, and the XML data statistics $xStats$. First, $pSchema$ is used to derive the corresponding relational schema (Section 4). This mapping is also used to translate $xStats$ into the corresponding statistics for the relational data (Section 5), as well as to translate individual queries in $xWkld$ into the corresponding relational queries in SQL (Section 4). The resulting rela-

tional schema and the statistics are taken as input by a relational optimizer to compute the expected cost of computing a query in the SQL workload derived as above; this cost is returned as the cost of the given *pSchema*.

Note that the algorithm does not restrict the kind of optimizer used (transformational or rule-based, linear or bushy, etc. [13]); although, obviously, the optimizer should reflect the actual costs in the target relational system. As a result, different database systems that use different optimizers and cost-models can easily be connected to LegoDB. Also note that physical design tools such as DB2 Advisor [21] and Microsoft's Tuning Wizard [2, 1] are complementary to LegoDB.

The first prototype of LegoDB implements a greedy search over inline and outline transformations, and experiments show this strategy is both efficient and effective in practice [5].

7 Discussion

In this paper, we discussed the main issues involved in building a colonial XML storage system based on adaptive shredding. We presented the architecture of LegoDB, described the implementation of its various components, and the underlying algorithms. The prototype implementation described here has been demonstrated at VLDB 2002 [4].

Initial experiments have shown that LegoDB is able to derive efficient relational configurations in a reasonable amount of time. We are currently investigating alternative search strategies that, in addition to inlining and outlining, also considers other transformations (e.g., union distribution, repetition split, etc). When additional transformations are considered, there is an explosion in the size search space, and a simple greedy strategy may miss interesting configurations.

In our prototype, we use the optimizer developed by Roy et al [18] to obtain cost estimates. In a *real* application, the optimizer should reflect the actual costs in the target relational system. We are currently investigating how to connect LegoDB to commercial RDBMS so that cost estimates can be obtained from these systems.

Acknowledgments

We would like to thank Jayant Haritsa, Maya Ramanath, Prasan Roy and Philip Bohannon for their invaluable contributions to the LegoDB project.

References

[1] S. Agrawal, S. Chaudhuri, and V. R. Narasayya. Automated selection of materialized views and indexes in SQL databases. In *Proc. of VLDB*, 2000. 115

[2] S. Agrawal, S. Chaudhuri, and V. R. Narasayya. Materialized view and index selection tool for microsoft sql server 2000. In *Proc. of SIGMOD*, 2001. 115

[3] S. Boag, D. Chamberlin, M. Fernandez, D. Florescu, J. Robie, J. Siméon, and M. Stefanescu. XQuery 1.0: An XML query language. W3C Working Draft, June 2001. 106

[4] P. Bohannon, J. Freire, J. Haritsa, M. Ramanath, P. Roy, and J. Siméon. Legodb: Customizing relational storage for xml documents. In *Proc. of VLDB*, 2002. 104, 115

[5] P. Bohannon, J. Freire, P. Roy, and J. Siméon. From XML schema to relations: A cost-based approach to XML storage. In *Proc. of ICDE*, 2002. 104, 108, 109, 113, 115

[6] M. J. Carey, J. Kiernan, J. Shanmugasundaram, E. J. Shekita, and S. N. Subramanian. XPERANTO: Middleware for publishing object-relational data as xml documents. In *Proc. of VLDB*, 2000. 104, 110

[7] A. Deutsch, M. Fernandez, and D. Suciu. Storing semi-structured data with STORED. In *Proc. of SIGMOD*, 1999. 104

[8] D. Draper, P. Fankhauser, M. Fernandez, A. Malhotra, K. Rose, M. Rys, J. Siméon, and P. Wadler. The XQuery 1.0 formal semantics, March 2002. W3C Working Draft. 109

[9] Excelon. `http://www.exceloncorp.com`. 104

[10] M. Fernandez, W. C. Tan, and D. Suciu. Silkroute: trading between relations and XML. *Computer Networks*, 33(1-6):723–745, 2000. 104, 106, 108, 110

[11] D. Florescu and D. Kossmann. A performance evaluation of alternative mapping schemes for storing XML in a relational database. Technical Report 3680, INRIA, 1999. 104

[12] J. Freire, J. Haritsa, M. Ramanath, P. Roy, and J. Siméon. Statix: Making XML count. In *Proc. of SIGMOD*, 2002. 111

[13] G. Graefe and W. J. McKenna. The volcano optimizer generator: Extensibility and efficient search. In *Proc. of ICDE*, 1993. 115

[14] IBM DB2 XML Extender. `http://www-3.ibm.com/software/data/db2/extenders/xmlext/library.html`. 104

[15] Oracle XML DB. `http://technet.oracle.com/tech/xml`. 104

[16] G. Piatetsky-Shapiro and C. Connell. Accurate estimation of the number of tuples satisfying a condition. In *Proc. of SIGMOD*, 1984. 112

[17] Raghu Ramakrishnan and Johannes Gehrke. *Database Management Systems*. McGraw-Hill, 2000. 112

[18] P. Roy, S. Seshadri, S. Sudarshan, and S. Bhobe. Efficient and extensible algorithms for multi query optimization. In *Proc. of SIGMOD*, 2000. 111, 115

[19] J. Shanmugasundaram, K. Tufte, G. He, C. Zhang, D. DeWitt, and J. Naughton. Relational databases for querying XML documents: Limitations and opportunities. In *Proc. of VLDB*, 1999. 104, 106, 113

[20] Tamino. `http://www.softwareag.com/tamino`. 104

[21] G. Valentin, M. Zuliani, D. C. Zilio, G. M. Lohman, and A. Skelley. Db2 advisor: An optimizer smart enough to recommend its own indexes. In *Proc. of ICDE*, 2000. 115

An Adaptable and Adjustable Mapping from XML Data to Tables in RDB

Wang Xiao-ling, Luan Jin-feng, and Dong Yi-sheng

Southeast University, Nanjing, China
{wxling,ysdong}@seu.edu.cn

Abstract. The purpose of this paper is to present the results of an initial study about automatically storing XML data in RDBMS. These are some research about storing given XML data, and most of these approaches based on pre-defined rules and heuristics, without considering application requirement and workload. As the result, the relational schema obtained for given XML is often not optimal for query performance. As a first step, this study was focused on providing an adaptable and adjustable strategy for modelling XML data in relational database systems. This AAM (Adjustable and Adaptable Method) takes advantage of the GA (Genetic Algorithm) to design data schema in RDBMS based on cost-driven approach, we implement and evaluate our approach. Experiments show that this method provides an adaptable and automatic mapping from XML data to physical tables in RDBMS with respect to different users and application requirement.

1 Introduction

Recently, XML has been a de facto standard for Internet data representation and exchange. With the wide use of XML, how to store, manage and query these useful self-describe data is an urgent task. Because the storage representation has a significant impact on query processing efficiency, the most important is how to store XML and what is the best way of storing XML data to improve query efficiency.

In general, there are different methods to store and query XML data, for example, file system, object-oriented database and RDB. Many projects [2][3][8] have proposed alternative strategies for storing XML documents. For example, XML data are stored in native XML database, such as Lore (a semi-structured data management by Stanford University) and Tamino (a native XML data management by Software AG). Today, most XML data are saved in a file system as text, but because a file system can't provide XPath and XQuery, and it must be parsed to DOM at each reading time, so it is exhausted when querying; On the other hand, it needs to provide index support for querying the XML data. Object-oriented model is more similar to XML model, and they all have some similar features, such as hierarchy, object, and other specific features, which might be useful for certain applications, but the current object-oriented database product is not mature enough to process complex queries on

S. Bressan et al. (Eds.): EEXTT and DIWeb 2002, LNCS 2590, pp. 117-130, 2003.

large databases, so OODB is not a good choice for XML storage. There are also many different ways to store XML data in RDBMS. There are some commercial RDB products (such as Oracle 8i or SQL Server 7), which support XML data storage, but most of these systems base on pre-defined rules and need user to decide how XML elements are stored in relational tables. And, there are another option based on data mining, to analyse the XML data and the expected query workload to building a mapping from XML data to physical tables, e.g., STORED [3]. Though there are also some works to compare these alternatives' performance and to answer which strategy is the best, as for we know, it is difficult to seek a "good" strategy for generic applications, and any strategy has specific application field, without considering application requirement and system workload.

Traditionally, database is a main tool for storing and querying data in a uniform data model. Among all database product, because of the mature of theories and applications, relation database is the most popular. The purpose of this paper is to present the results of an initial study about storing XML data in relational database with an adaptable and adjustable method--AAM. In this work, we do not involve any analysis of the XML data.

On the other hand, from the view of traditional database, it is very important to get an optimal schema in order to improve the query performance, because the storage representation has a significant impact on query processing efficiency. But it is very difficult to choose the good schema, because the number of feasible schema for given XML data may be very large, that is the search space is usually very large, exhaustive search for the optimal schema is impractical. Until now, most database modelling is based on designers' intuition and experience, and there is no good method to store XML data in relation database with optimal schema for specific application and workload. As the result, we design a program of "Genetic Algorithm" to choose a best mapping according to the query workload. Genetic algorithm got inspiration from Darwin's evolution theory and has been used to solve some problems, in which size of the search or complexity of the function precludes traditional approaches and the objective may change over time [7]. By combining evolution computing with computer science and technology, there are many inter-cross research field, for example evolving neural net, evolving modelling, evolving computation, evolving learning, evolving software, evolving hardware, and so on. We get some new novel ideas from evolution theory and use it for XML data schema design and development in RDBMS.

The contribution of this paper mainly includes:

1. In a relational database, the key issues about automatically generation of the mapping from an XML document to physical schema are studied, and we apply Genetic Algorithm method to search the optimal schema for given XML data and queries workload in order to avoid exhaustive searching.
2. Automatically turning the physical structure according to the user's query workload. It is very important to take query performance into consideration at the time of designing physical schema in relation database. It is also helpful to consider workload for storing XML data in relation database. In this paper, we focus on designing table structures for giving XML data and query requirement based on cost-driven approach.

3. This method can be a strategy to create many material views for various applications or users, referring to their different query requirements. The method presented in this paper not only is a schema selection approach for storing XML data in a relational database, and it also can be a method to model database based on cost model.

This paper is organized as the following: Section 2 is about related work about XML data Storage by others; Section 3 gives a description of Genetic Algorithm; Section 4 is to present our AAM algorithm in detail; Experiments and evaluation are in Section 5; In section 6, conclusions and future work are presented.

2 Related Work

Several projects have investigated strategies for storing XML data. Some research is to investigate native XML data management system, such as Lore [11][12], which is a special purpose database system that exploits features of OEM, the semi-structured data model. Such native XML systems are designed for specific XML data model, it starts from scratch to design and implement all data management functions, including data storage, data segment, data security, query rewrite, query optimal, and so on. Until now, most storage strategy is based on RDBMS, because RDBMS provide some mature function for data management.

There are three mapping methods to store XML documents in a RDBMS [2], such as edge approach, binary approach and universal approach. All methods are about how to map edge in XML schema graph to tables in relational database. In edge approach, it is store all edges of the graph that represents an XML document in a single table. The edge table has the following structure: Edge (source, ordinal, name target), where source and target objects' oids, the label of the edge and an ordinal number are recorded. In binary approach, all edges with the same label in edge approach are grouped into one table. Each *Binary* table has the following structure: Binary (source, ordinal, flag, target). The third approach is a single *Universal* table to store all the edges. This *Universal* table corresponds to the result of a full outer join of all *Binary* tables. The structure of the *Universal* table is as follows: Universal (source, $ordinal_{n1}$, $flag_{n1}$, $target_{n1}$, $ordinal_{n2}$, $flag_{n2}$, $target_{n2}$, …, $ordinal_{nk}$, $flag_{nk}$, $target_{nk}$). Former research has shown the *Universal* table has many fields which are set to *nul*l, and it also has a great deal of redundancy. We observe that these three methods are all hard-coded *ad hoc* heuristics, such as universal approach has only one table and edge approach has many little tables, which only have few rows. So, they should some hybrid map approaches, rather than only three approaches. In this paper, another different approach is exploiting based on cost-driven.

Except for manual design data schema in RDBMS for XML data, there is few works to create data schema automatically. STORED [3] is the most successful attempt to get data schema with data mining technique. Data mining is a good foundation for the STORED generation algorithm, they try to search for more complex patterns and attempt to cover most of the data, but not all. They test STORED by exploiting the regularities and the results shows that the generated STORED queries can cover a large percentage of the data. The key method for

generate data schema in STORED is extended data mining techniques to semi-structured data [4].

We have briefly described these alternatives by others. In this paper, we study how to generate data schema from XML data automatically, rather than manually; our approach can store all XML data without loss, on this point, this method is different STORED [3], which store data partially.

3 Genetic Algorithm

Though there are some researches about XML storage and query, as for we know, few studies adapt genetic algorithm and evolutionary computation as alternative strategies. We get some new novel ideas from evolution theory and use it for XML data schema design and development in RDBMS.

Design XML storage in RDB is a process of design and implement physical model to satisfy users query requirements. How to design an effective physical model of RDB involves a continuous interplay between what we want to achieve and how we want to achieve it. Design defined by Gero is "Design is a search process in which a satisfactory design solution is produced from a number of alternatives [5]." That is to say, how to design the RDB mapping schema for XML data is a series of comparing and choosing the best component satisfied by the users' query requirement or storage demands, and this process can be simulate by genetic algorithms.

Genetic Algorithm (GA) was first described by John Holland [7]. The underlying technique of generic algorithm is Evolutionary Computing (EC). EC is a technique to solve optimal problems, where the gene values are combined to form a binary string of fixed length. The advantage of this approach is that the programs can evolve to be far larger and more sophisticated than members of the initial population. This means that potential solutions are not restricted by the generality of the representation used in the initial population. In these years, there are many developments in evolutionary computing, such as Genetic Programming (GP), evolutionary programming (EP) and evolution strategy (ES). Genetic algorithms provide an available and robust method of self-adaptable, self-learning, self-organization [6], so it is a good method to take advantage of the merits of evolutionary computation and genetic algorithm to implement our goal in order to adapt to query workload's change. In this paper, we introduce GA to design a dynamic, effective and efficient mapping from XML data to tables in RDBMS.

There are mainly three operators in GA: propagate (deliver merit to offspring), crossover (create new population by cross over two individual in old population society) and mutation (create new feature by change one or more features from an old individual). Crossover operator involves the exchange of portions between two individuals; Mutation operator involves changing the attributes of an individual. By mutating and crossing over, the GA is essentially experimenting with new solutions while preserving potentially valuable interim results. The genetic algorithms paradigm allows the several of potentially huge problem spaces in a parallel and efficient manner.

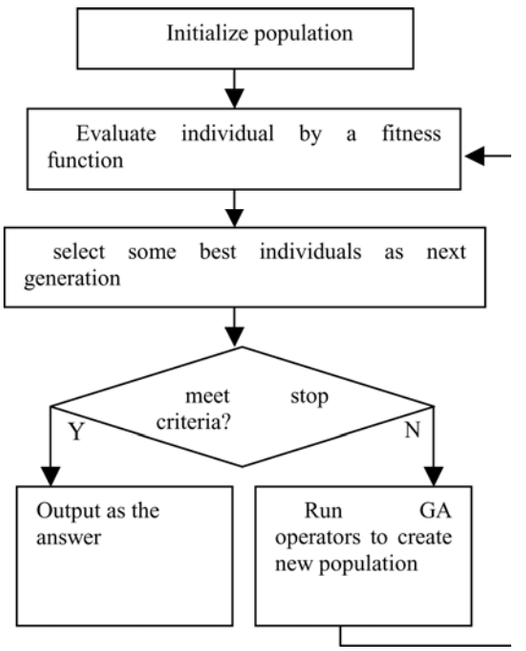


Fig. 1. The procedure of GA

The procedure of GA is as Figure 1. In GA, "encoding" and "fitness function " are the keys to solve problem. "Encoding" is the base of algorithm and it reflects the problem's character. "Fitness function" is to control the search space, and good fitness will cause to get optimal solution as soon as possible.

In section 4, we will focus on how to encoding the problem of the mapping from XML data to tables and how to define fitness function to resolve the mapping problem from many alternative approaches.

4 Adaptable Mapping from XML Data to Relational Tables

Though there are some strategies for XML storage as we discuss in section 2, for example, file system, OODB and native XML storage, in this section, we study the mapping from XML data to tables in Relational Database. Because there are many important merits about RDBMS: firstly, RDBMS is mature in all database products from the aspect of theory. Secondly, RDB is the most widely use in applications and most of useful data exist in RDB. As the result, our goal is how to map XML data into tables in relational in order to take advantage of merits of RDBMS.

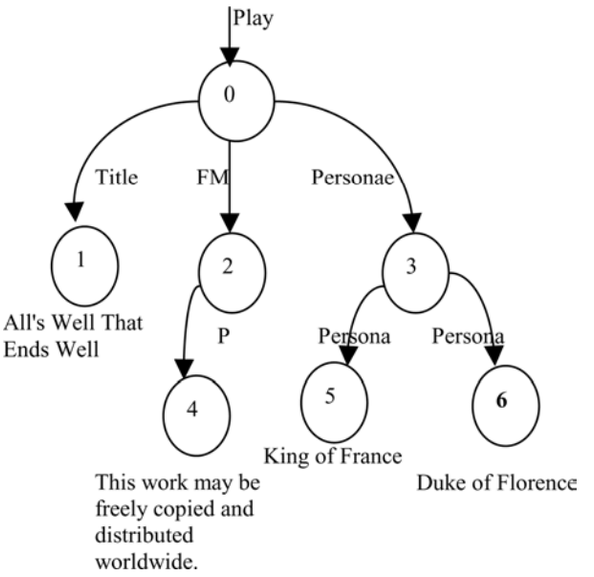


Fig. 2. XML data graph

4.1 Model

Mostly, an XML document can be represented as an ordered and labelled directed graph. Edge is the label in XML document and the node is labelled with the *oid* of the XML object. Our AAM method depends on this graph model (Figure 2).

The first question is how to model our problem. Given an XML data, our goal is to generate an efficient and effective relational mapping schema. From traditional database view, a good schema usually includes minimum disk spaces and minimum query cost, so the criteria depend on application fields. Even for the same application, with the time on, the application will focus on different part of those data, so the query model is changing and it needs to re-design the data schema to reduce the weighted cost of those queried. In this paper, we present a method to get the best relational mapping schema according to the current query model, when the query model changes, the algorithm will be triggered again to recalculate the mapping scheme, if the result schema is not equal to the current schema in use, then replace the mapping schema with the new, and then transfers all the data. We now describe how to generate the relational mapping for given XML data instance D. Depending on the application, some goals may reduce object splits and add redundant storage in multiple relations, or, on the contrary, increasing object redundancy to improve query evaluation [3]. The goal is to find the best mapping from all possible approaches, that is to say, find the minimum cost mapping: the cost includes memory cost and the query workload. The optimization problem is trying to find a minimum cost *C* to store and query XML data *D*.

Definition: Given XML documents D, and a query model Q(D), find the optimal relational schema S for D, such that cost C(D) is minimal.

This problem can be reduced into the problem of query optimization problem. The mapping problem is NP-hard in the size of the data, and it had proved [3].

Theorem 1: The problem of computing an optimal storage mapping M is NP-hard in the size of XML data D.

The cost C(D) includes disk space M(D) and query cost Q(D).

$$C(D) = M(D) + Q(D)$$

In this formula, M(D) is decided by the space-cost function, and it is calculated by the disk space of all tables. Q(D) is the cost of query-cost function, which is got by calculate "select" cost and "join" cost for a query model. This query model includes a series of queries and its probability, which is got by user input at first or automatically generated statistic data when the system is running. In a query model Q, there are two aspects, the first is about the select queries S and their probabilities P_s; the second is about the join queries J and their related probabilities P_j. So, for each mapping approach, this query model's query cost Q(D) is a sum of select cost and join cost. For a given query model Q(D) = $\{(S_1,P_{s1}),(S_2,P_{s2})..(S_m,P_{sm}),(J_1,P_{j1}),(J_2,P_{j2}),\ldots(J_n,P_{jn})\}$, the cost Q(D) is calculated by the following formula.

$$Q(D) = \sum_{i=1}^{m} S_i * P_{s_i} + \sum_{k=1}^{n} J_k * P_{j_k}$$

4.2 Encoding XML Schema for GA

An evolutionary system normally includes an algorithmic calculation of an individual's fitness, allied with a number of genetic operators, used to produce members of the next generation. Hence, in such a system, three aspects deserve special attention: the fitness function, the genetic operators and the representation technique. In this section, we focus on how to design the fitness function and the encoding representation.

Firstly, it is a key to design a bit string to represent the question. In order to evolve schema, the represent must be flexible and has ability to represent all possible mapping approaches. When dealing with schema mapping, the ability to evolve the inherent complexity of schema, as well as their constituent elements, is of great value.

In order to show the bit string about how XML data is mapped into relational tables in each scheme, we will use an XML example from shakespeare.1.10.xml[10] to illustrate our algorithm.

In Figure 3, each node is an object id and its label, for example, "play:0" represents that this object id is 0 and "play" is the edge to access oid 0; "Title:10" means that oid 10 can be accessed by label "Title" from oid 4.

Given a schema as Figure 2, to design bit string represent for GA is as following. For each bit in this string, when it is set to 1, it means that this edge will create a table to store its data and/or its offspring (when its offspring have created tables, the ancestor will not store data for offspring). Though there are seventeen edges in this schema, sixteen bit is enough, because the edge "play" to root oid 0 will create a table

anyway. When all bits are set 1, all tables are built. Under this condition, the system will create sixteen tables for each edge and another table for edge ”play” as root. This schema is very simple, but it will spend much more cost for some queries about join two or more table.

1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

On the other hand, when all bits are set to 0, none of separate tables are built, except a table is built for the edge “play”, which is the root. Under this schema, all offspring are stored in “play” table, which is similar to *universal* table [2]. This mapping schema is useful for some queries, but it will spend much disk space to store NULL, because this schema is non-normalize and there is much redundancy [2].

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

As we shown, this bit string can represent traditional mapping, such as *Universal* Table. Other possible of bit string can be interpreted to another approach of mapping. For example, when bit string is 1111111111111001, there are fourteen tables are created except “Line” table and “Stagedir” table. All attributes about “Line” and “Stagedir” are attributes of their parent “Speech” table, so data of “Line” and “Stagedir” are stored in parent table.

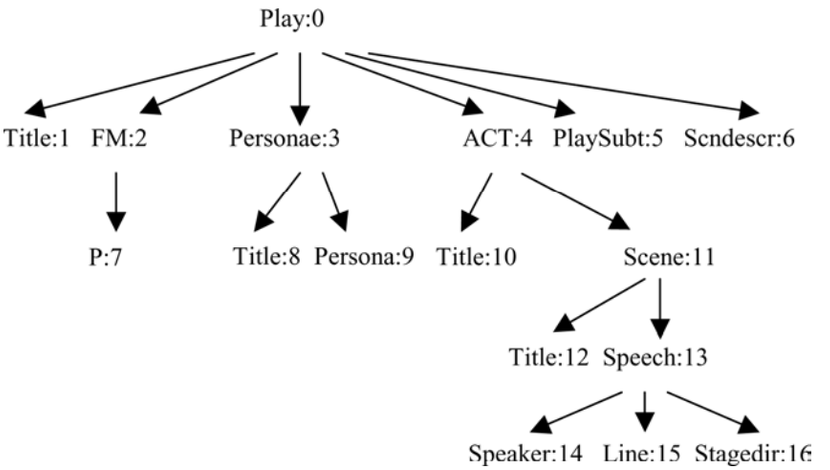


Fig. 3. One part of XML Schema in Shakespeare Play

4.3 Generic Algorithm for Mapping

We have shown that mapping can be encoded in bit string, when one bit is set to 1, an table will create; when one bit is set to 0, no table will create for it, its attribute and value will be stored in its parent or ancestor. Under this bit string, in order to evolve schema, an initial population will be created randomly, the members are tested and their fitness is evaluated, then a new generation will be created from the old. In the next generation consists of the best of the previous generation, a collection of new individuals produced by genetic operators (crossover, mutation and others). Fitness is criteria to decide if individuals are good enough to keep, if not, the weaker individual will be replaced by better. When any individual arrive the fitness threshold set by use,

system will terminate at this best answer. System also will exit when a certain number of generations completed, even if the best answer will not got. In this section, we first introduce our fitness function and how to apply GA for mapping XML Schema to tables in RDBMS.

Under this encoding, any mapping will correspond to a bit string and any bit string will create some tables related to this mapping. The fitness function is defined as the cost of the mapping, so our goal is to search the entire generation to find out the minimum cost. Supposed there are q bit in a bit string, the total cost of this bit string is as following.

$$C(D) = \sum_{l=1}^{q} F_l * R_l + \sum_{i=1}^{m} S_i * P_{s_i} + \sum_{k=1}^{n} J_k * P_{j_k}$$

Where D is a given XML data instance;

q is the number of this bit string;

m is the number of the "select" operation in a query model;

n is the number of the "join" operation in the query model;

F is the number of the fields in a table T_l.

R is the number of the rows in a table T_l.

$\sum_{l=1}^{q} F_l * R_l$ is the total memory cost of this mapping model.

$\sum_{i=1}^{m} S_i * P_{s_i} + \sum_{k=1}^{n} J_k * P_{j_k}$ is the total query cost of this model for a specific query model.

As for query model, before tables create, users decide what SQL operation mostly run on which objects, such as which objects will often be retrieved, or which two or more objects will be combined to search. Under this guidance, tables are generated by this method. When system is in use, this query model will be got from users SQL operations. We will not introduce this point in this paper, and we focus on how to use this query model for design an adaptable and adjustable mapping from XML to tables.

The initial population can be given a certain profile by altering the various parameters controlling the size of population, the maximum size of generation and the probabilities of occurrence of the operators. The steps applied to evolving initial individuals are as follows.

```
ALGORITHM: AAM Generator

INPUT: XML Schema and query model

OUTPUT: Bit string of the minimum cost

METHOD:

Step 1: Encode XML Schema with the approach we
presented in section 4.2;
```

```
Step 2: Initialization,Create an initial generation at
random;

Step 3: Repeat step 4 and step 5 until stopping
criteria is met.

Step 4: Using the fitness function we defined in
Section 4.2 to evaluate individual in this generation,
and select a number of population as the next
generation.

Step 5: Applying the genetic operators (crossover,
mutation or propagate) to create new population.
```

In this algorithm, stopping criteria is the number of the generation. We have implemented this system, at most cases, the best string of sixteen bit can get within twenty generation. We will discuss the experiments in the Section 5.

5 Performance Experiments and Results

We carried out a series of performance experiments in order to study the tradeoffs of the alternative mapping schemes in an RDBMS. As an experimental platform, we use a SQL Server 2000 installed on a PC with 933 MHz processors and 128 MB of main memory.

Firstly, we study how to decide parameters for GA. We must control crossover probability and mutation probability to get the answer as soon as possible. The following two experiments are for the selection of parameters.

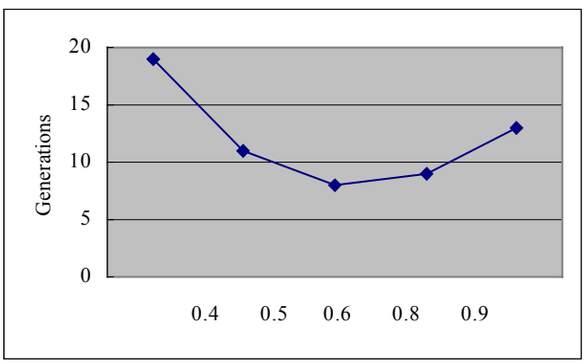


Fig. 4. Answer for different Pc

When the Pm (the probability of mutation) is set to 0.1, Figure 3 shows the best answer for different Pc (the probability of crossover) will first get at which generation. Our experiments show algorithm will get answer quickly when Pc is from 0.6 to 0.8. When the probability of crossover is smaller or larger, the rapid of convergence will fall.

Another tests are to decide the best value of Pm for our model.

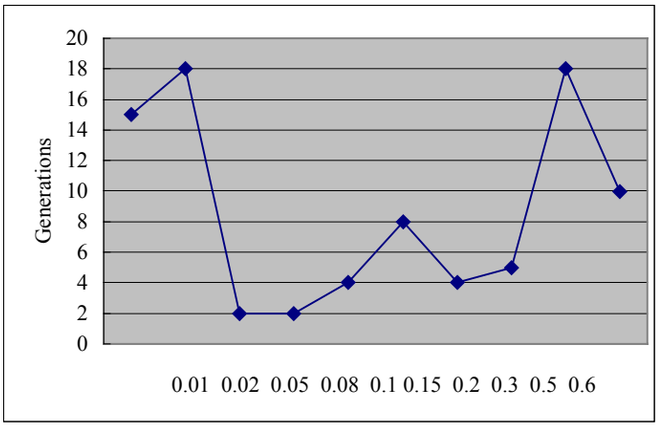


Fig. 5. Answer for different Pm

The best probability of the mutation is 0.08. When much more bits run mutation, it will cause the vibration of candidate answers, so it can't reach the best answer as soon as possible. When few bits run mutation, it can't bring more new bit strings to the population, so the rapid of convergence is slow. For our model, the best probability of mutation is about 0.08.

The next experiment is to show the answer for a specific query model. In this test, the GA parameters we choose are that the probability of crossover is 0.7 and the probability of mutation is 0.08. The two values are got from Fig. 3 and Fig. 4.

Our algorithm creates seventeen tables stored in SQL server 2000: PLAY, PLAY_TITLE, PLAY_ACT, PLAY_PERSONAE, PLAY_SCNDESCR, PLAY_FM, PLAY_PLAYSUBT, FM_P, ACT_SCENE, ACT_TITLE, PERSONAE_PERSONA, PERSONAE_TITLE, SCENE_SPEECH, SCENE_TITLE, SPEECH_LINE, SPEECH_SPEAKER, SPEECH_STAGEDIR. Table "PLAY" is a root table. Other tables are for each edge in Fig. 2, such as "PLAY_TITLE" is for the edge from "PLAY" to "TITLE". The data in SQL Server about this table is Table 1. In our experiment, we store data value inline, for example, we store "The Tragedy of King Lear" in the field of "Title".

Table 1. "PLAY_TITLE" table

Id	Title	Play_ID
1	The Tragedy of King Lear	0

Edge tables or Binary tables are simple mapping to store XML data, though these method avoid data redundance, when there are some SQL operations, which involve two or more tables, it is exhausted to run "join" operations on these tables. For example, if many queries are about "what is the speaker KENT say in which ACTION". Then, three tables of "SPEECH_SPEAKER", "SPEECH_LINE", "ACT_TITLE" will often combine by "join" operations. So, the cost for each query is

high and waste. For this query model, we run it by our AAM approach, the answer bit string (1111111101111001) is got at the second generation. The answer mapping shows that no table is for "SPEECH_SPEAKER", "ACT_TITLE" and "SPEECH_LINE", and the data of edge "speaker" and "line" are stored in the their ancestor "PLAY_ACT" table. Though our memory cost is a little of higher, it will decrease query cost extremely. In addition, it is flexible and adaptable with different query model and memory model. So, it can be a wizard to guide user to design physical model for XML data in RDBMS.

The next experiment is to show the answer for a specific query model. In this test, the GA parameters are the probability of crossover is 0.7 and the probability of mutation is 0.08 coming from last experiment. In this test, we change the number of bits for the same query model, in order to test the efficiency of our algorithm. The results are shown in Table 2. Our query model is that the "join" probability of the third and the forth table is 0.90. Because the third and forth are often "join", they should be "join" and no tables are created separately, their data should be stored in their ancestor table. So, all results must be 11001111…, it means that the third and the forth bit assigns to 0 and others to 1. In our experiments, results can be got no more than 50 generations, mostly results occur within ten generation, with the increase of the number of bits, it is hard to find the result easily, after several decades, result is achieved. In general, result can be got within several mill-seconds.

Table 2. Experiment Results

Number of bits	Time (ms)	Generation where result occur	Result in Hex	Result in Binary
4	1	1	0xc	1100
6	2	1	ox33	110011
8	2	1	oxcf	11001111
9	3	1	ox19f	110011111
12	3	5	0xcff	110011111111
16	4	22	oxcffff	1100111111111111111

6 Conclusion

With the growing importance of XML as a standard to represent data in the World Wide Web, there has been a lot of effort on storing XML documents. Our focus in this paper are getting an effective and efficient approach to store XML with sophisticate mapping, rather than simple mapping, such as edge approach, which stores all of the parent-child relationships in an edge table. Evolutionary computation and genetic algorithms provide a generic and robust method to achieve optimal schema from a large search space in a reasonable time. We focus on how to design mapping systems automatically with genetic algorithm. The potential advantages of this approach are many – easily create a specific mapping for a kind of users, getting the minimum cost in the global. This method can be a wizard of physical model for XML document in RDBMS.

We studied approaches about mapping schemes to tables in a relational database. This study was only a first step towards finding the best way to store XML data. We present a cost-driven approach to generate an optimal relational schema for a given XML data and query requirement. Our results can be used as a basis to develop more sophisticated mapping schemes. We have implemented our prototype. Initial experiments show that it is a good approach to design physical model for XML documents. The results are satisfied and acceptable.

There are still some works need to study, such as how to design mapping for XML documents without XML Schema or DTD, how to reconstruct XML document from tables. We will address these studies in the future work.

Acknowledgements

Thank Prof Lu HongJun for the helpful discussion. This study was supported by Southeast--Nanrui Foundation.

References

[1] F. Tian, D. J. DeWitt, J.j. Chen and C. Zhang. The Design and Performance Evaluation of Alternative XML Storage Strategies. Technical report, CS Dept., Universiy of Wisconsin, 2000. Available at http://www.cs.wisc.edu/niagara/papers/vldb00XML.pdf.

[2] D. Florescu, D. Kossmann. Storing and Querying XML Data using an RDMBS. Data Engineering , September 1999 Vol. 22 No. 3. p27-34.

[3] A. Deutsch, M. Fernandez, and D. Suciu. Storing semistructured data with STORED. In *Proc. of ACM SIGMO*D, Philadelphia, PN, 1999.

[4] K. Wang and H.q. Liu. Discovering typical structures of documents: a road map approach. In ACM SIGIR Conferences on Research and Development in Information Retrieval, August 1998.

[5] R. Q. Lu, J. Zhi, G. Chen, Ontology-oriented Requirement Analysis, Journal of Software, 2000,11(8).

[6] Kenneth De Jong. Evolving in a changing world. International Workshop on Evolutionary Computation. April, 2000.

[7] J. H. Holland. Adaptation in Natural and Artifical Systems. University of Michigan Press. Ann Arbor, MI, 1975.

[8] D. Florescu and D. Kossmann. A performance evaluation of alternative mapping schemes for storing XML data in a relational database. Technical Report, INRIA, France, 1999.

[9] J. Shanmugasundaram et al. Relational databases for querying XML documents: Limitations and opportunities. In *Proc. of VLD*B, Edinburgh, Scotland, 1999.

[10] Shakespeare.1.10.xml.http://metalab.unc.edu/bosak/xml/eg.

[11] Abiteboul, S., Quass, D., McHugh, J., Widom, J. and Wiener, J. The Lorel query language for semistructured data. International Journal on Digital Libraries, 1(1):68--88, April 1997.
[12] Roy, G., Narayanan, S., Suresh, V. and Hector, G. Proximity Search in Databases. In Proceedings of the 24th VLDB Conference, 1998.

Efficient Structure Oriented Storage of XML Documents Using ORDBMS

Alexander Kuckelberg[1] and Ralph Krieger[2]

[1] Chair of Railway Studies and Transport Economics, RWTH Aachen
Mies-van-der-Rohe-Str. 1, D-52056 Aachen, Germany
kuckelberg@via.rwth-aachen.de
http://www.via.rwth-aachen.de
[2] Institute of Information Systems (i5), RWTH Aachen
krieger@post.rwth-aachen.de

Abstract. In this paper we will present different storage approaches for XML documents, the document centered, the data and the structure centered approach. We will then focus on the structure centered approach and will introduce an abstract view on XML documents using tree graphs. To make these tree graphs persistent different storage techniques are mentioned and evaluated concerning the creation and retrieval of complete documents. Measurement results are shown and shortly discussed for creation and retrieval of different (complete) XML documents. Moreover we will shortly introduce the partial mapping extension, which helps to optimize the generic structure based storage approach for specific documents whose structure is known in advance.
The results presented come from the ongoing implementation of a high performance generic document server with an analytic decision support agent.

1 Introduction

The grown acceptance of XML documents implied the development of additional tools, management and transformation mechanisms and XML related topics.

One aspect of this field is the persistence of XML structured data respectively XML documents. A common approach is based on database systems. Current standard database systems – e.g. DB2 (IBM), the Oracle databases, Sybase, Informix – usually implement the relational data model with varying object oriented extensions.

Storing XML data within these (relational) databases entails the problem of differing data models and concepts. While the (Object-) Relational Databases ((O)RDB) realize a relational, set oriented approach XML data follows a hierarchically approach (from a syntactical point of view).

Moreover the varying structure of XML documents with respect to element attributes, their domain, element values and domains, numbers of encapsulated elements contradicts the concept of fixed predefined structures of (O)RDB management systems ((O)RDBMS).

S. Bressan et al. (Eds.): EEXTT and DIWeb 2002, LNCS 2590, pp. 131–143, 2003.

Varying solutions have been developed and evaluated to map flexibly structured XML documents to fixed (OR)RDB schemas. They can be roughly categorized as document oriented, data oriented and structure oriented. These categories are described in more detail in Section 2.

The structure oriented storage approach it is focused in our work due to its general validity. Additionally it retains the structure of XML documents. With sufficient extensions current extensible databases support the structure oriented storage approaches in a good manner with respect to data indexing and retrieval, too. This paper presents our approach to structure oriented XML document storage, the evaluation of varying data indexes and how this storage approach can be combined with data oriented approaches.

The paper is structured as follows: Section 2 introduces the different variants of storage approaches for XML documents, Section 3 considers implementation aspects of the structure oriented storage approach including the foreign key (Section 3.2), the depth first (Section 3.3) and the SICF (3.4) indexing and how the data oriented approach can be integrated (Section 3.5). Section 4 presents and discusses measurement results for some usage scenarios and Section 5 finally concludes.

2 Storage Approaches

Different approaches to store XML documents within databases have been tested, used and implemented, e.g. [5, 4, 2, 13, 14]. The most common data model is the relational one and most commercial relational database systems offer (build-in) extensions to handle XML documents. A very common one is storing and probably indexing XML documents *as a whole* (document centric) whereas the transformation of document structures into database schemas is less frequently implemented.

2.1 Document Centered Storage

The document centered storage approach handles the document as an atomic unit. The database considers the XML document therefore as a single large object, storing it as a (big) binary or character large object.

Consequently the retrieval of (any part of) the stored XML document requires – without any further consideration of e.g. partial access to large objects – accessing the whole attribute value. Accesses to the whole or at least to major parts of the document have a very good performance.

But retrieving XML data partially implies the access to the whole data as well. Parts of the document can only be returned after the document has been parsed and the requested data have been extracted. This additional – and useless – workload leads to decreasing performance for partial document retrievals.

Querying and evaluating predicates against small parts of the document data is another problem. The drawback of accessing the whole document can partially be equalized by indexing methods like full text indexing, keyword extraction or

knowledge retrieval techniques whose data is handled within the database as well and used for query processing and answering. Therefor the efficiency of this storage approach – with respect to partial document retrieval of querying – strongly depends on the additional indexing techniques.

This document centered storage approach is implemented by almost all major (O)RDBMS and optimized for XML document management [9, 8, 11, 10]. In principle, this approach – large object storage with sufficient indexing data extraction methodologies – is suitable for nearly any kind of document like Postscript files, Word documents or images, not only those containing XML data.

2.2 Data Centered Storage

The data centered storage approach focuses the content of a XML document. The key aspect influencing the mapping between XML documents and database schemas is the transformation of XML data structure into database structures. This implies that the structure of XML documents directly imply the mapping, which so becomes document dependent.

The approach can be roughly described: Define (or detect) frequent structures of expected XML documents and map them to database schema, define rules how elements, attribute values and element content fits to the schema, how relations have to be combined to retrieve (parts of) the original XML document etc.

Document Type Definitions (DTDs) are an example of these structure definitions. They can be used as given knowledge about XML document structures. Starting with DTDs, required relations can be generated based on algorithms. Fitting mapping rules for splitting XML documents or rebuilding them from relation data can be generated as well. Different algorithms have been proposed and evaluated, e.g. [2, 4, 5]. Some are sufficient for specific scenarios, some imply strong data redundancy, others are very join intensive when recreating a document. Depending on the concrete implementation approaches are suited for certain tasks and application areas.

Conceptually this approach has a major drawback: the structure of expected or accepted XML documents must be known in advance – or determined on runtime whenever a XML document is inserted – to defined the required mapping rules and schemas.

Another drawback is that usually several relations are involved when storing or retrieving a document. Therefor several read/write or join operations with multiple random access operations are required. Moreover data is often stored redundantly in several relations. The redundant data might be used to decrease the number of join operations but this needs a good knowledge about the system and its algorithms.

One advantage of this approach is that it supports querying documents efficiently because the mapped data can be indexed easily by build-in database indexes.

Retrieving small parts of a document is efficient, too, but the retrieval of large parts or of the whole document leads to bad retrieval performances.

2.3 Structure Centered Storage

The structure centered storage approach tries to overcome the major drawback of the data centered approach, the required knowledge of document structures in advance. The structure centered approach is motivated by the limited number of XML concepts, e.g. elements, attributes, element content etc. [1].

These core XML concepts are mapped to relations and tuples. This abstraction makes the structure centered approach generally valid and enables the handling of arbitrary XML documents in an uniform manner. Principally each concept is mapped to one relation, the relations refer each other with foreign key relationships. In this way relations are created for elements – probably with self references –, the data content of elements – probably distributed on different relations which are optimized for one specific data type –, the attributes of elements and attributes realizing cross references.

This approach considers the syntactical aspects of XML documents and can easily be realized using existing tools (e.g. DOM parser[12]).

From a more abstract point of view the XML document is considered as a tree graph where nodes have a type and a value. With respect to storage and retrieval of typed tree graphs a linear order of nodes can be defined, – e.g. in a depth first manner – which can be used e.g. by generic B-trees. The usage of such a tree index helps to decrease the number of required joins, e.g. for retrieving the whole document or parts of it. But primarily the joining and retrieving the original document can be done sequentially, no random access operations to document data are required anymore.

The evaluation of predicates against documents or parts of it, e.g. the data content of specific elements, can use built-in database indexes as well.

Our work focuses this technique and extends the pure structure oriented storage approach by integrating data oriented approaches to increase creation and retrieval performance.

3 Structure Centered Storage Techniques

The structure centered storage approach is an approach valid for arbitrary XML documents. Standard build-in database indexes can be used easily to index XML data. The most important question is how to store the typed tree graph of a XML document such that most operations like storing or retrieving documents, evaluating predicates on the documents data, modifying document structures etc. can be performed efficiently.

3.1 The Data Model

XML document are mapped to typed tree graphs. All nodes belong to one single class. A node has a label, a type, a content and a list of child nodes. There are no cycles. The following notation is formally used: If $\mathcal{N}$ is the set of graph nodes representing the XML document, a single node $\nu \in \mathcal{N}$

is a tuple $\nu = (t, l, c, n)$, where t is the type of the node ($t \in \mathcal{T}$, $\mathcal{T} = \{ELEM, ATTR, TXT, IDREF, \cdots\}$, l is the node label, c is the node content and $n = \{\nu_1 \cdots \nu_m\}$ ($m \in \mathbb{N}, \nu_i \in \mathcal{N}$ for $i = 1, \cdots, m$) is the set of successor nodes (see example below).

These graph nodes can easily be mapped into (O)RDBMS data models. Two aspects are very important when implementing the storage: the domain of the content c and how child node sets are realized.

As long as all nodes are kept in one single (database) tuple the content data type must be the most general data type for all possible node content values, which means that casts must exist from all possible content data types into the most general data type, e.g. `LVarChar`. The usage of one general data type for all values is not advisable due to redundancy etc.

This problem can be solved by distributing the tuples on several relations – with the same structure from a conceptual point of view but with different data types for the content values – depending on node position, type or label. These ideas are also used for the partial storage in Section 3.5.

Three realizations of child node sets have been tested. The first realizes node references with traditional key/foreign key references, which is supported by database with primary and secondary access structures. The second and third realization use specific index values which represent the position of a node within the tree graph and which can be totally ordered and therefore indexed by the built-in database indexes. The depth first index value (Section 3.3) aggregates minimum and maximum ranges, the SICF index value (Section 3.4) uses simple continued fractions as node position identifiers. Both index structures allow the definition of total ordering functions, with which generic indexes (B-trees) can be configured. This will speed up the access performance. Finally truncating the trees and storing the remaining tree separately leads to reduced tree depths and partial tree storage (Section 3.5).

The following XML document extract is used throughout the following chapters:

```
...  <person PNr="23884-23">
            <name><last>Smith</last><first>Jon</first></name>
            <birthday>03.10.1973</birthday>
            <title>Dr.</title><title>Prof.</title>
            <address IDREF="1234"/>
     </person> ...
```

3.2 Foreign Key

The foreign key storage approach is a straightforward approach, which fits plain relational databases. Each tree node $\nu \in \mathcal{N}$ is mapped to a tuple with an unique identifier and a foreign key reference to the parent node.

This approach is simple and can be implemented by almost any relational database. Documents can easily be modified, e.g. moving sub trees by just updating one parent reference. A huge problem is the retrieval of documents, numerous self-joins (with random accesses to secondary storage media) are required.

The nodes should be stored in a depth first manner. This might allow retrieving documents with sequential scans instead of random accesses. The foreign key approach can consider this by assigning identifier in increasing depth first order to the nodes but this would destroy the simple modification ability.

3.3 Depth First

A depth first (DF) index value is associated to each tree graph node representing its position. The plain DF value aggregates a minimum and a maximum value, extended versions include e.g. depth values.

The two values (min/max) are determined in the following way: When traversing the tree in a depth first manner a counter is increased each time another node is (re-) visited. If a node is visited the first time its minimum DF value is set to the current counter value. When all child nodes have been visited the maximum DF value is again set to the current counter value. Additionally the depth $(0 \cdots)$ is stored within each DF value as well. The DF-coding of the XML extract is shown in Figure 1.

Relationships of arbitrary nodes can easily be determined using their DF values: A node ν is contained within the subtree of node μ if the minimum of ν (denoted as ν_{min}) is greater than μ_{min} and $\nu_{max} < \mu_{max}$. A node ν has been traversed before node μ if μ is contained in the subtree of ν or $\nu_{max} < \mu_{min}$.

Using these relationships the graph nodes can be ordered totally. Finally this comparison can be implemented and can be used e.g. by generic database B-trees (or derivations) to set up primary access methods.

With this approach joins are avoided when retrieving (parts of) documents because a linear scan is possible. This obviously increases the retrieval performance (Chapter 4.2). Predicates can easily be defined and implemented, e.g. a *contained* predicate searching subtrees of nodes for a certain node, value or type etc. When predicates are implemented as user defined routines they are

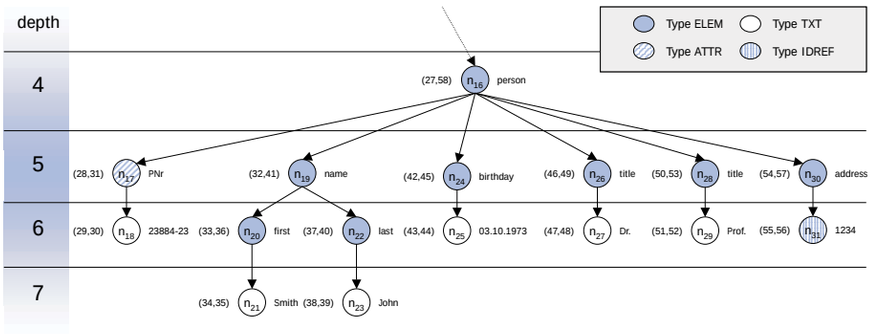


Fig. 1. Example of a Depth First indexed tree (extract)

executed within the database kernel and therefor increases query evaluation performance.

Unfortunately document modifications become more complex because probably a lot of DF values have to be updated.

3.4 Simple Continued Fraction

A simple continued fraction (SICF) σ is a fraction of the form

$$\sigma = \cfrac{1}{q_k + \cfrac{1}{\ddots_{\ldots q_2 + \frac{1}{q_1}}}} (= \frac{N_k}{D_k} = \sigma_k = \frac{N}{D})$$

$(q_i \in \mathbb{N}, i = 1, \cdots, k)$ where $q_1, \cdots, q_k$ are called the *partial quotients* of σ (notation: $< q_1, \ldots, q_k >$) of positive numbers, the *partial quotient sequence of* σ. Moreover $q_1 > 1 \Rightarrow 0 < \sigma < 1$ holds [3]. SICF values are used in the same way as DF values to index tree graph nodes.

Partial quotient sequences uniquely determine fractions and vice versa which enable the indexing of nodes: The root node gets a *seed value* $s \in \mathbb{N}, s > 1$, its SICF value is $\frac{1}{s}$ $(\equiv < s >)$. If a node ν with SICF value $< q_1, \cdots, q_m >$ has n (ordered) child nodes $\nu_1, \cdots \nu_n$, than the SICF value for each child node is determined by $< q_1, \cdots, q_m, i >$ (see Figure 2).

The SICF values enables the definition of user defined routines and predicates similar to DF values. Therefore using the extensibility of current databases tree indexes can be set up with SICF values as well because functions comparing two SICF values can be defined easily.

The comparison is based on partial quotient sequence comparison: If one sequence is the prefix of the other, then the fraction associated to the shorter sequence is less than the other. Otherwise the first partial quotient q_i where

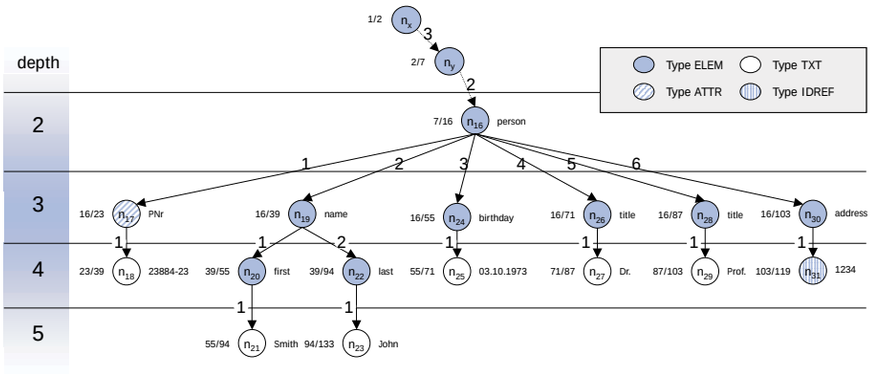


Fig. 2. Example of a SICF indexed tree (extract)

both sequences differ determines the node order. The fraction with the lower quotient is less than the other one. In the same way predicates can be defined, e.g. *is_successor_of*, *is_sibling_of* or *has_parent*.

One intention when using the SICF values was the observation that performance od database operations usually depend on I/O performance to secondary storage space (hard disk) whereas the workload of the CPU is not the most important performance factor. The SICF approach shifts this workload partially to CPU operations. Hopefully this total system will have a better performance due to a better workload balance between CPU and I/O operations.

Advantages and disadvantages will be comparable to the DF value approach. With SICF values more detailed predicates are possible and computable, e.g. the real difference of two child node positions, the direct computation of sibling SICF values etc. SICF values and the comparison functions are used together with build-in database indexes. Due to detailed predicates and the indexing functionality a good retrieval and query performance was expected for this approach.

3.5 Partial Data Structured Storage

As mentioned before the storage techniques can be extended by graph truncating rules. The rules define conditions depending on node positions, type or content within a graph tree. Whenever a condition is satisfied, the subtree of the corresponding node is truncated and stored in a document or data centered manner. In this way, the generally valid structure oriented storage approach can be mixed with structure specific storage aspects.

With partial data structured storage tuples representing specific nodes can be clustered within specialized relations. The major advantage of this partial mapping is that the overall storage approach remains generally valid but it moreover allows optimization whenever the structural knowledge is available in advance.

4 Measurements

Some measurement results are presented and shortly discussed in this chapter. The first measurement series was based on one large XML document representing an extract of the US baseball league. The scalability should be evaluated within the series.

Another measurement series used generated XML documents with a fixed depth of 1 and a varying number of total nodes to test tree width dependency. Similar experiments with increasing tree depth have also been performed but they are not represented here: The SICF approach fails if a certain depth is reached (huge size of numerators and denominators) and therefore comparability is missing. Even if this points out to be a major drawback of the SICF approach we didn't find any 'real world' XML document, for which we reached these limitations.

In the following the inserting – called creation – and the retrieval of a document are described. The measurement results of modification and detailed query

evaluation are currently evaluated but preliminary measurement results show a superior performance of the SICF approach over the DF approach and a very dominating performance advantage compared to the foreign key approach. The measurements have been performed on a Pentium III/800 machine with 256MB main memory and fast SCSI hard discs with mirrored databases. As database the Informix Dynamic Server 9.22 was used, which was extended by data types and user defined routines [7, 6].

4.1 Creating a Document

The first series of measurements focused the dependency from the number of documents handled. For this purpose, the large baseball XML document was inserted several times and performance times were measured.

In Figure 3 the overall creation time for different numbers of documents can be seen. Obviously the creation time remains approximately constant, which implies that all approaches are well scalably with respect to the number of documents inserted. One remarkable observation is that the foreign key approach is the fastest, which might come from the fact, that only build-in and well optimized functionalities are used here whereas the user-defined computations for DF and SICF values might be less optimized or integrated into the database kernel. Moreover the SICF approach requires more time than the depth first (DF) approach, probably because the computation of SICF values is more complex than the corresponding DF value computation.

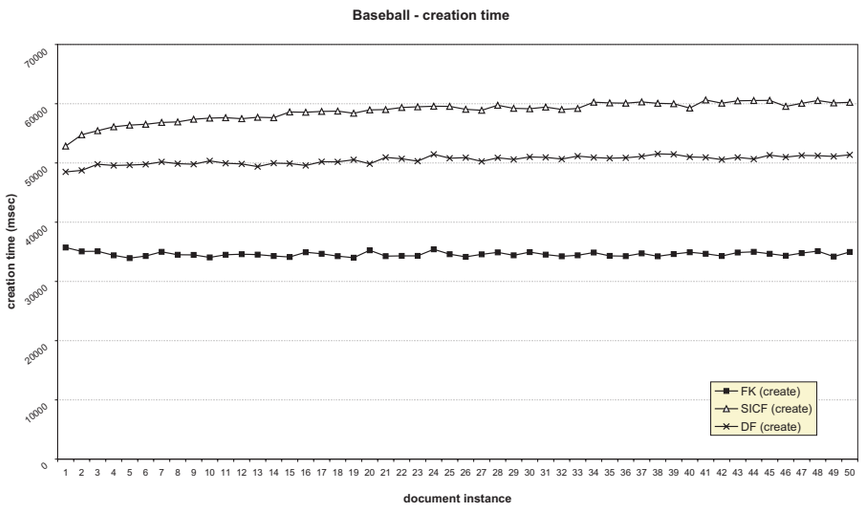


Fig. 3. Creation time for multiple instances of the baseball document

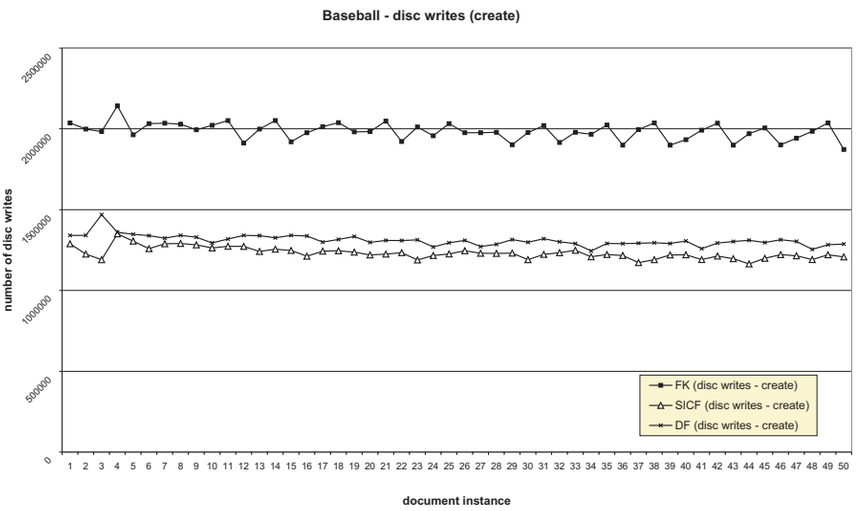


Fig. 4. Number of disc writes when creating multiple instances of the baseball document

Another interesting aspect is visualized by Figure 4. The number of disc write operations which have been performed to create the documents are shown. Obviously the FK approach needs the most write accesses coming from primary and secondary access structure maintenance whereas DF as well as SICF values are only occurring as leaf nodes of primary access structures. The mirrors layout of the databases could be one reason for the still faster overall storage time of the FK approach.

The creation time measurements for generated documents of fixed depth and varying width showed an approximately linear behavior of all three approaches. The DF approach was slightly faster than the other two.

4.2 Retrieving a Document

Similar to the creation the retrieval of the baseball document has been measured. In accordance to the creation process the retrieval time is shown in Figure 5.

It is obviously that the foreign key approach is not sufficient for fast document retrieval, especially when the complete document has to be reconstructed from multiple tuples. In contrast to (probably a lot of) random accesses to relation tuples with the FK approach the DF as well as the SICF approach allow a linear scan of relations because the nodes are already clustered in a depth first manner due to the user defined comparison routine. But nevertheless all approaches are stable with respect to the number of documents stored.

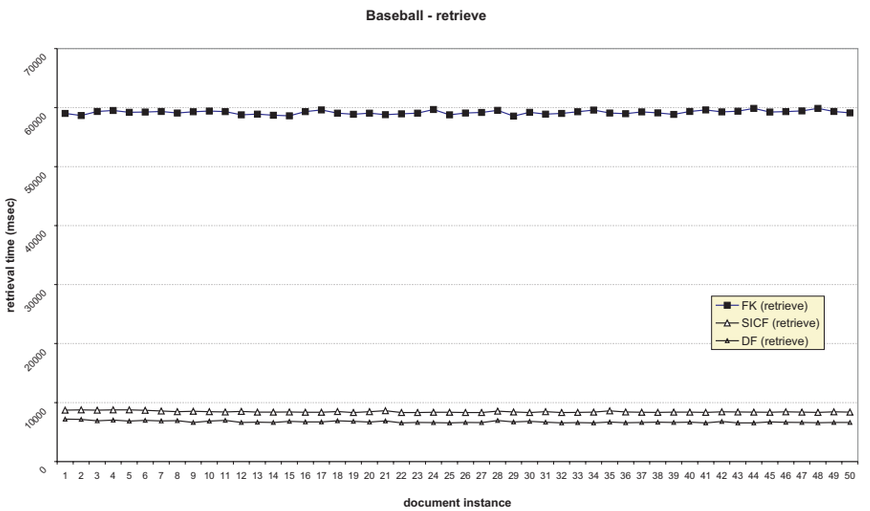


Fig. 5. Retrieval time for multiple instances of the baseball document

In contrast comparing the retrieval time for the different sized generated documents leads to bad scalability for the FK approach whereas the DF and SICF pproach behaves much better as shown in Figure 6.

As a consequence the FK approach might be a good choice for any archiving system with none or just a few accesses to the documents (*backup of documents*).

4.3 Partial Retrieval and Queries

The ongoing experiments with DF and SICF indexed documents shows a significant better performance of the SICF approach whenever documents are queried in detail and not only the retrieval performance of (parts of) the documents are measured. To confirm these measurement results more measurements should be done soon.

5 Conclusion and Outlook

In this paper we presented different storage approaches for XML documents and focused on the structure centered one. For this we introduced an abstract view on the XML document using tree graphs. To make these tree graphs persistent different storage techniques have been mentioned and evaluated concerning the creation and retrieval of complete documents.

Due to ongoing evaluation of measurements and the limited size of this paper the modification of documents and the mapping was not shown in detailed diagrams but the measurements showed a very good behavior of the SICF approach

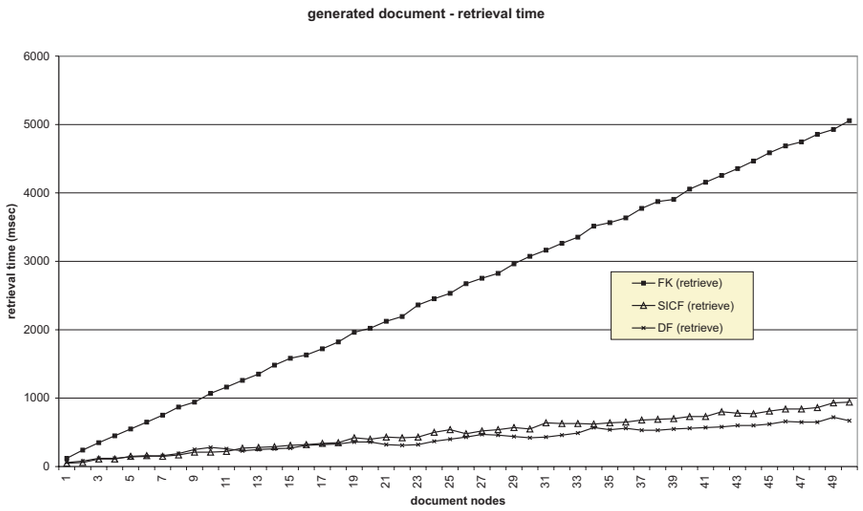


Fig. 6. Retrieval time for the generated documents (nodes in hundreds)

in conjunction with specialized user defined routines used as predicates within database queries.

References

[1] T. Shimura, M. Yoshikawa, S. Uemura, *Storage and Retrieval of XML Documents Using Object-Relational Databases*. 134

[2] J. Shanmugasundaram, K. Tufte, G.He, C. Zhang, D. DeWitt, J. Naughton. *Relational databases for querying XML documents: Limitations and opportunities*, VDLB Conference, Edinburgh, Scotland, 1999. 132, 133

[3] P. Zezula, Fausto Rabitti, *Object Store with navigation acceleration*, Oktober 1992. 137

[4] D. Florescu, D. Kossmann, *A performance evaluation of alternative mapping schemes for storing XML data in a relational database*, Technical report, INRIA, Rocquencourt, Mai 1999. 132, 133

[5] A. Schmidt, M. Kersten, M. Windhouwer, F. Waas, *Efficient Relational Storage and Retrieval of XML Documents*, http://citeseer.nj.nec.com/-schmidt00efficient.html. 132, 133

[6] Online documentation, *Informix Dynamic Server 2000*, `http://www.informix.com/answers/english/pids92.htm`. 139

[7] Online documentation, *DataBlade Modules and DataBlade Developer's Kit*, `http://www.informix.com/answers/english/pdbm.htm`. 139

[8] *The Object Translator Solution*, Feature Descriptions, `http://www.informix.com/idn-secure/webtools/ot/features.html`. 133

[9] IBM DB2 XML Extender, *An end-to-end solution for storing and retrieving XML documents*, http://www-4.ibm.com/software/data/db2/extenders/xmlext/xmlextbroch.pdf. 133
[10] *Oracle Homepage* http://www.oracle.com. 133
[11] *Using XML with the Sybase Adaptive Server SQL Databases*, A Technical Whitepaper (1999), http://www.sybase.com. 133
[12] *Document Object Model (DOM)*, November 2000, http://www.w3.org/DOM/. 134
[13] R. Bourret, *XML and Databases*, September, 1999, http://www.rpbourret.com/xml/XMLAndDatabases.htm. 132
[14] D. D. Kha, M. Yoshikawa, S. Uemura, *An XML Indexing Structure with Relative Region Coordinate*, http://citeseer.nj.nec.com/391959.html. 132

Assessing XML Data Management with XMark

Albrecht Schmidt[1], Florian Waas[2], Martin Kersten[1], Michael J. Carey[3], Ioana Manolescu[4], and Ralph Busse[5]

[1] CWI, Kruislaan 413, 1090 GB Amsterdam, The Netherlands
{albrecht.schmidt,martin.kersten}@cwi.nl
[2] Microsoft Corporation, Redmond (WA), USA
florianw@microsoft.com
[3] BEA Systems, Inc., USA
mcarey@bea.com
[4] INRIA-Rocquencourt, 78153 Le Chesnay Cedex, France
ioana.manolescu@inria.fr
[5] FHG-IPSI, Dolivostr. 15, 64293 Darmstadt, Germany
busse@ipsi.fraunhofer.de

1 Motivation

We discuss some of the experiences we gathered during the development and deployment of XMark, a tool to assess the infrastructure and performance of XML Data Management Systems. Since the appearance of the first XML database prototypes in research institutions and development labs, topics like validation, performance evaluation and optimization of XML query processors have received significant interest. The XMark benchmark follows a tradition in database research and provides a framework to assess the abilities and performance of XML processing system: it helps users to see how a query component integrates into an application and how it copes with a variety of query types that are typically encountered in real-world scenarios. To this end, XMark offers an application scenario and a set of queries; each query is intended to challenge a particular aspect of the query processor like the performance of full-text search combined with structural information or joins. Furthermore, we have designed and made available a benchmark document generator that allows for efficient generation of databases of different sizes ranging from small to very large. In short, XMark attempts to cover the major aspects of XML query processing ranging from small to large document and from textual queries to data analysis and *ad hoc* queries.

2 Some Lessons Learned

During the experiments and evaluation that we conducted the following points of interest came up: (1) The physical XML mapping has a far-reaching influence on the complexity of query plans. Each mapping favors certain types of queries and enables efficient execution plans for them. No mapping was able to outperform the others across the board in our experiments. (2) The complexity of query plans is often aggravated by information-loss during translation from the declarative

S. Bressan et al. (Eds.): EEXTT and DIWeb 2002, LNCS 2590, pp. 144–145, 2003.

high-level query language to the low-level execution algebra. There often appears to be a semantic gap between the two – at least in the implementations we inspected. Thus, cost-based query optimizers tend to consider search spaces that are larger than necessary. Further research might lead to query algebras that reduce the gap. (3) Meta-data access can be a dominant factor in query execution especially in simple lookup queries with small result sizes. It is possible that rather complex relationships have to be extracted from the database's meta data store. (4) Schema information often enables better database schema design and is also useful in query optimization since it introduces syntactic and semantic constraints that can guide the search for a good execution plan, reduce storage requirements or enable clustering. (5) An XML query engine is usually only one part of a large system and has to integrate well with the other components.

More and complementary information especially about results obtained from running the benchmark on several platforms can be found at the places listed in the References below.

References

[1] A. Schmidt, M. Kersten, D. Florescu, M. Carey, I. Manolescu, and F. Waas. The XML Store Benchmark Project, 2000. http://www.xml-benchmark.org.

[2] A. Schmidt, F. Waas, M. Kersten, M. Carey, I. Manolescu, and R. Busse. XMark: A Benchmark for XML Data Management. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, pages 974–985, Hong Kong, China, August 2002.

[3] A. Schmidt, F. Waas, M. Kersten, D. Florescu, I. Manolescu, M. Carey, and R. Busse. The XML Benchmark Project. Technical Report INS-R0103, CWI, Amsterdam, The Netherlands, April 2001.

The XOO7 Benchmark

Stéphane Bressan[1], Mong Li Lee[1], Ying Guang Li[1],
Zoé Lacroix[2], and Ullas Nambiar[2]

[1]National University of Singapore
{steph,leeml,liyg}@comp.nus.edu.sg
[2]Arizona State University
{zoe.lacroix,mallu}@asu.edu

1 Introduction

As XML becomes the standard for electronic data interchange, benchmarks are needed to provide a comparative performance analysis of XML Management Systems (XMLMS). Typically a benchmark should adhere to four criteria: relevance, portability, scalability and simplicity [1]. The data structure of a benchmark for XML must be complex enough to capture the characteristics of XML data representation. Data sets should be in various sizes. Benchmark queries should only be defined with the primitives of the language.

XML models present similarities with object-oriented data models. While XML is able to handle semi-structured data, it supports most of the features of complex object models. Classes, methods and inheritance are not defined in XML but classes can be expressed through element types and attributes. Thus in developing a benchmark for XML, we decided to use the well-established OO7 benchmark [3] designed for object-oriented database management system as a starting point. The XOO7 benchmark, an XML version of the OO7 benchmark, is a single-user based benchmark for XMLMS that focuses on the query processing aspect of XML.

The DTD and data set of XOO7 are directly obtained by mapping the OO7 schema and data set to XML [4]. OO7 does not model any specific application, but it intends to capture the characteristics of an object-oriented database. Additionally, in order to cater for the document centric view of XML, we extended the document object of OO7 to contain sub-elements mixed with text data. Thus, the *Document* element provides for a liberal use of free-form text that is "marked up" with elements. Therefore the XOO7 data set can capture all the characteristics of typical XML database applications. We provide a parameterized program to generate XML databases of various sizes and characteristics.

The XOO7 benchmark also extends and modifies the eight OO7 queries with a set of twenty-three queries. XOO7 provides relational, document and navigational queries that are specific and critical for XML database applications. The queries test the primitive features and each query covers only a few features. XOO7 queries are defined to express the requirements published by the W3C XML Query Language working group. XOO7 queries are therefore supported by most XMLMS, which makes them very portable. Users can choose a subset of queries: data centric, document or navigational, to test on the features required in their applications [2], [5].

S. Bressan et al. (Eds.): EEXTT and DIWeb 2002, LNCS 2590, pp. 146-147, 2003.

XOO7 has been implemented and used to evaluate two XML-enabled management systems: LORE, XENA and two native-XML management systems: Kweelt, DOM-XPath[1]. The experiment results and the analysis can be found in [5].

Up-to-date information about the XOO7 benchmark can be found at the web site: http://www.comp.nus.edu.sg/~ebh/XOO7.html.

References

[1] Gray, J.: The Benchmark Handbook: for Database and Transaction Processing Systems, 2nd Edition, Morgan Kaufmann Publishers, Inc., 1993.

[2] Bressan, S., Lee, M. L., Li, Y. G., Lacroix, Z., Nambiar, U.: The XOO7 XML Management System Benchmark. National University of Singapore CS Department Technical Report TR21/00, November 2001.

[3] Carey, M. J., DeWitt, D. J., Naughton, J. F.: The OO7 Benchmark. In: Proceedings of the ACM SIGMOD International Conference on Management of Data, Washington D.C., 1993, pp. 12-21.

[4] Nambiar, U., Lacroix, Z., Bressan, S., Lee, M. L., Li, Y. G.: XML Benchmarks Put to the Test. In: Proceedings of the Third International Conference on Information Integration and Web-based Applications and Services, Linz, Austria, September 2001.

[5] Nambiar, U., Lacroix, Z., Bressan, S., Lee, M. L., Li, Y. G.: Efficient XML Data Management: An Analysis. Proceedings of the Third International Conference on Electronic Commerce and Web Technologies, Aix en Provence, France, September 2002.

[1] The name of the product is not published at the request of the developers.

Multi-user Evaluation of XML Data Management Systems with XMach-1

Timo Böhme and Erhard Rahm

University of Leipzig, Germany
{boehme,rahm}@informatik.uni-leipzig.de
http://dbs.uni-leipzig.de

Abstract. XMach-1 was the first XML data management benchmark designed for general applicability [1]. It is still the only benchmark supporting a multi-user performance evaluation of XML database systems. After a brief review of XMach-1 we summarize three additionally proposed benchmarks (XMark, XOO7, Mbench) and provide a comparison between these benchmarks. We then present experiences and performance results from evaluating XML database systems with XMach-1.

1 Introduction

Current XML database systems exhibit considerable differences in their architectural foundations, concepts and functionality. When choosing a system for a specific application scenario these aspects and the resulting performance should be taken into consideration. The intention of XML database benchmarks is the comprehensive and realistic evaluation of XML database systems in order to allow for a performance comparison of them.

Several papers about storing XML in databases contain performance measurements based on self defined benchmarks [2, 3]. These studies commonly lack a detailed benchmark description, have only a few very specific operations tailored to the corresponding subject of the paper and are therefore not suitable for a general comparison.

The growing demand for well defined XML data management benchmarks led to the specification of various benchmarks in the last two years. The applicability of a benchmark depends on how close the benchmark simulates the application domain in question. To find the most appropriate benchmark it is necessary to carefully study the specifications of each of them. We therefore provide a comparison between four benchmarks highlighting their key features and distinctive features.

The rest of this paper is organized as follows. In the next section, we give an overview of the first general XML database benchmark, XMach-1. Section 3 briefly introduces three further XML database benchmarks: XMark, XOO7 and Mbench. We then compare the four benchmarks with respect to key features. In Section 5, we present experiences and results from evaluating XML database systems with XMach-1.

S. Bressan et al. (Eds.): EEXTT and DIWeb 2002, LNCS 2590, pp. 148-159, 2003.

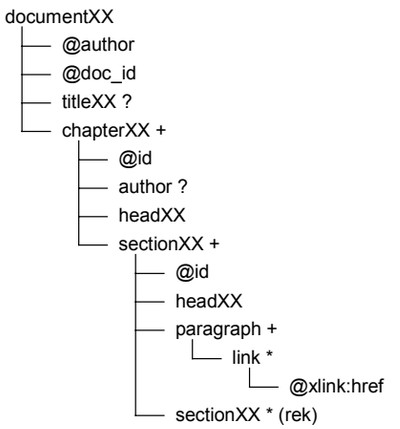


Fig. 1. Node hierarchy of XMach-1 text document

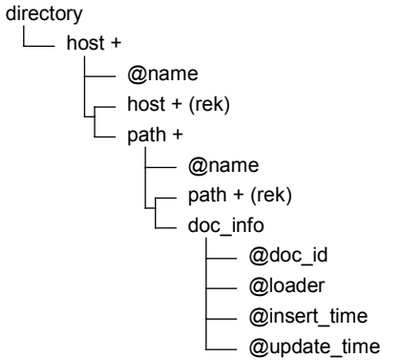


Fig. 2. Node hierarchy of XMach-1 metadata document

2 XMach-1 – An Overview

XMach-1 was developed by us at the University of Leipzig in 2000 and published at the beginning of 2001 [1]. It was thus the first published XML database benchmark with general applicability.

Three objectives are central for the design of the benchmark: scalability, multi-user processing and evaluation of the entire data management system. XMach-1 is based on a web application in order to model a typical use case of an XML data management system. The system architecture consists of four parts: the XML database, application servers, loaders and browser clients. The database contains a directory structure and XML documents that are assumed to be loaded from various data sources in the internet by loader programs (e.g. robots crawling the web or a registration tool where

web-site authors can add documents to the database). Every document has a unique URL which is maintained together with document metadata in the directory structure. The application servers run a web (HTTP) server and other middleware components to support processing of the XML documents and to interact with the backend database.

The XML database contains of both types of documents: document-centric and data-centric. The largest part consists of document-centric XML data which mimic text documents such as books or essays in structure and content. These documents are synthetically produced by a parameterizable generator. In order to achieve realistic results when storing and querying text contents, text is generated from the 10,000 most frequent English words, using a distribution corresponding to natural language text. The documents vary in size (2-100 KB) as well as in structure (flat and deep element hierarchy). Fig. 1 shows the element and attribute hierarchy of these documents.

Table 1. Operations defined in XMach-1

ID	Description	Comment
Q1	Get document with URL X.	Reconstruction of complex structured document with ordering preserved.
Q2	Get doc_id from documents containing phrase X in a paragraph element.	Tests full-text retrieval capabilities.
Q3	Start with first chapter element and recursively follow first section element. Return last section elements.	Simulates navigating a document tree using sequential operators.
Q4	For a document with doc_id X return flat list of head elements which are children of section elements.	Restructuring operation simulating creation of a table of contents.
Q5	Get document name (last path element in directory structure) from all documents which are below a given URL fragment.	Operation on structured unordered data.
Q6	Get doc_id and id of parent element of author element with content X.	Selection using element content.
Q7	Get doc_id from documents which are referenced at least X times.	Tests group by and count functionality.
Q8	Get doc_id from the last X updated documents having an author attribute	Needs count, sort, join and existential operations and accesses metadata.
M1	Insert new document.	Tests insert performance for complex document with activated indices.
M2	Delete document with doc_id X.	Tests deletion performance for complex document with activated indices.
M3	Update name and update_time attributes for document with doc_id X.	Tests efficiency of update operations on attribute values.

The data-centric type is represented by a document containing metadata about the other documents such as URL, name, insert time and update time. All data in this document is stored in attributes (no mixed content) and the order of element siblings

is free. Compared to structured data in relational databases it shows some semi-structured properties like variable path length using recursive elements or optional attributes. The structure of this document is depicted in Fig. 2.

A distinctive feature of XMach-1 is support of a large number of document schemas with 2-100 documents per schema. This allows us to test a database system's ability to cope with a variable number of different element types and to test query execution across multiple schemas. Additionally, the benchmark supports schema-based as well as schema-less document storage.

The database contains at least 1000 text documents and can be scaled by increasing the number of documents by a factor of 10, 100, 1000, etc. The metadata document scales proportionally with the number of text documents. Ratios such as the number of documents per schema, number of authors per document etc. remain constant when scaling the database.

The XMach-1 workload mix consists of 8 query operations and 3 update operations which are described in Table 1. They cover a wide range of processing features like querying complete complex structured documents, full-text retrieval, navigational queries, queries using sorting and grouping operators etc. A formal specification of the operations in XQuery[1] syntax can be found in [4]. Update operations cover inserting and deleting of documents as well as changing attribute values. Despite the missing data manipulation language for update operations, we consider it as essential especially for multi-user performance to evaluate workloads with both queries and updates.

Since XMach-1 is a multi-user benchmark the primary performance metric is throughput measured in Xqps (XML queries per second). This value is calculated from the workload mix which defines firm ratios for each operation. The mix emphasizes the retrieval of complete documents whereas update operations have only a minor share of 2%. Nevertheless the latter one can have a significant impact on the concurrent execution of queries requiring access to the most recent data.

3 Further XML Database Benchmarks

After the specification of XMach-1 was published a number of additional XML database benchmarks were proposed. In this section we briefly introduce the benchmarks XMark, XOO7 and Mbench.

XMark. This benchmark was developed at the National Research Institute for Mathematics and Computer Science (CWI) of the Netherlands and made public in the middle of 2001 [5]. The benchmark focuses on the performance evaluation of the query processor which is reflected by the large number of specified operations. The guidelines of the benchmark design are discussed in [6].

The benchmark data is modeled after an internet auction database. It consists of a number of facts having a firm structure with data-centric aspects. However some document-centric features are introduced by the inclusion of textual descriptions. The

[1] http://www.w3.org/TR/xquery/

complete data is contained within a single document. For most of the element types the sibling order is free.

XMark's operations are made up of 20 queries. No update operations are specified. The queries are designed to capture different aspects of the query processing. Some queries are functional similar to test certain features of the query optimizer. In [7] seven systems were evaluated using XMark. It was shown that no system was able to outperform the others in all disciplines. Rather each physical XML mapping favors certain types of queries for which efficient execution plans could be generated.

XOO7. This benchmark was published [8] shortly after XMark and is a development from the National University of Singapore. It is derived from the object oriented database benchmark OO7 [9] with small changes in the data structure and additional operation types to better meet XML usage patterns.

In contrast to XMach-1 or XMark no specific application domain is modeled by the data. It is rather based on a generic description of complex objects using component-of relationships. This regular and fixed structure having all values stored in attributes exhibits a strong data-centric character. Similar to XMark some document-centric aspects are included using document tags with mixed content. Likewise the database is represented by a single document.

The benchmark only considers query operations grouped by the authors into relational queries, navigational queries and document queries. It defines a total of 23 operations including some newly added queries, especially in the document queries group. The evaluation focus is on the performance of the query processor in single user mode on a central server (1 computer).

Mbench. One of the latest additions to the family of XML database benchmarks is the Michigan Benchmark developed at the University of Michigan [10]. In contrast to its predecessors it is designed as a micro-benchmark aiming to evaluate the cost of individual pieces of core query functionality. Therefore it abstracts from specific application-level approaches defining only well-controlled data access patterns. This kind of benchmark restricts the operation execution to single user mode.

The benchmark data set is a synthetic structure created to simulate different XML data characteristics and to enable operations with predictable costs. Like XMark and XOO7 only one document covers the complete data. The main data structure consists of only one element which is nested with a carefully chosen fanout at every level. With an element hierarchy of 16 and a fixed fanout for each level most of the elements are placed at the deepest level. A second element is used to add intra-document references. The first element contains a number of attributes which can be used to select a defined number of elements within the database. With only two element types and the large number of attributes the data has clearly data-centric properties. Document-centric features are also represented since every element has mixed content and the element sibling order is relevant.

In order to meet the requirements of a micro-benchmark Mbench defines many (56) operations which are grouped into the categories selection queries, value-based join queries, pointer-based join queries, aggregate queries and updates. Within each group, often queries differ only with respect to a specific feature such as selectivity to measure its influence on query performance. Since the data set consists only of two element types typical navigational operations using element names are missing.

4 Benchmark Comparison

In this section we compare the introduced benchmarks with respect to key features such as application focus, evaluation scope, multi-user support, database and workload characteristics, etc. The comparison is intended to help choosing among the benchmarks for a particular evaluation purpose or application domain. Table 2 summarizes the main features which we will now discuss.

The high flexibility of XML leads to vastly different data structures, data types and operations in differents applications. The benchmarks try to accommodate typical characteristics from both document-centric and data-centric XML usage but with different focus. XMach-1 emphasizes the document-centric aspect the most while the other benchmarks focus on data-centric properties with a fixed database structure or a high number of attributes. Mbench is less data-centric (more document-centric) than XMark and XOO7 since each element has textual content.

Table 2. Comparison of XML database benchmarks

	XMach-1	XMark	XOO7	Mbench
main data focus	document-centric	data-centric	data-centric	data-centric
evaluation scope	DBMS	query processor	query processor	core query operators
# user	multi-user	single-user	single-user	single-user
# server	>=1	1	1	1
# documents	10^n (n>=3)	1	1	1
# schemas	#documents/20	1	1	1
# element types	4 * #schemas + 7	74	9	2
DB size	16 KB * #documents	10 MB – 10 GB	ca. 4 MB – 1 GB	50 MB * 10^n (n=1,2,3,4)
#nodes/KB	10	18	67	12
# queries	8	20	23	49
# update op.	3	0	0	7

A fundamental difference between the benchmarks lies in their evaluation scope. With its concept of evaluating the entire database system in multi-user mode XMach-1 covers the user view on the system as a whole. Therefore all components of the database system like query processing, caching, locking, logging etc. are included in the evaluation. The other benchmarks restrict themselves to the evaluation of the query processor in single-user mode to determine the performance for specific queries. XMark and XOO7 evaluate fairly complex queries stressing various features of the query language, while Mbench uses a higher number of smaller operations to systematically evaluate core functions (operators) of the query processor.

All XML data of a database can either be in a single document or spread across several documents. XMach-1 uses many smaller documents with a mean size of 16 KB. This allows easy scalability of the database and gives flexibility to the database system for data allocation, locking, caching etc. The other benchmarks require

the whole database be a single document. This is a significant restriction for some current XML database systems performing document-level locking etc. and would make it difficult to use these benchmarks for multi-user processing.

Since one of the strengths of XML lies in the flexible schema handling it should be natural for an XML database to easily handle multiple schemas. However this feature is only tested in XMach-1. The other benchmarks use a single fixed schema. As a result the number of element types remains unchanged for different database sizes. With its very small number of element types Mbench leads to artificial storage patterns in systems with a element-determined database organization such as some XML-to-relational mapping approaches.

Each of the benchmarks supports a parameterized scaling of the database from a few megabytes to some gigabytes. However the performance for loading the database, querying etc. is not only determined by the size but also by the structural complexity of the data. To give a rough indicator for this we have determined the number of XML nodes[2] per kilobyte data. As indicated in Table 2 for each benchmark this ratio is invariant w.r.t. the database size. XOO7 has by far the highest ratio which stresses its data-centric focus. The large share of textual content in Mbench is also reflected in its comparatively low value.

The differences in evaluation scope can also be seen in the number of query operations. XMach-1 evaluating the whole database system in multi-user mode specifies only a smaller number of complex queries since throughput is the primary metric. XMark and XOO7 having their focus on the query processor use twice as many query operations to capture most query capabilities. Mbench has even more operations to evaluate distinct parts of the core functions of the query processor. Only two benchmarks, XMach-1 and Mbench, consider update operations although they can impact performance substantially.

The comparison shows that both XMach-1 and Mbench have a clear focus. XMach-1 is targeted to evaluating entire database systems in multi-user mode using document-centric data whereas Mbench focuses on evaluating the core operators of the query processor in single user mode. XMark and XOO7 are similar in many respects. Their key differences come from the different schema characteritics. Here XMark has some advantages by supporting a rich structure as well as a more realistic text usage than XOO7.

5 XMach-1 – Experiences and Results

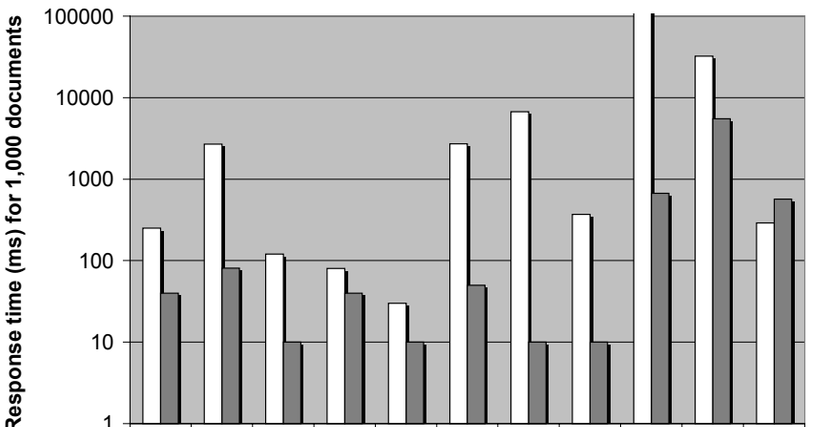
Since the first implementation of XMach-1 in early 2001 we used it to evaluate several XML database systems and subsequent versions of them. We discuss some of our experiences and present some performance results to indicate the performance achieved by current systems.

We started with the evaluation of native XML database systems. Their increased XML functionality over XML-enabled relational DBMSs made it easier to implement XMach-1. Still these products were rather new on the market and exhibited significant limitations, especially w.r.t. full-text indexing and multi-user processing. Prob-

[2] An XML node is either an element or an attributes.

lems were unacceptably long full-text index generation times, lacking support for phrase searches and for indexing across multiple schemas. In multi-user mode we observed substantial locking bottlenecks due to document-level locking leading to very high query response times during parallel writes. Locking at the document level would obviously be completely unacceptable for a database with a single document only. Other locking problems were caused by the index updates for inserting or deleting documents. Some systems were unable to support more than 20 concurrent clients and crashed. In [11] we discuss further problem areas for the first versions of XML database systems.

Most of the issues were resolved in subsequent versions of the systems leading to improved performance. This is exemplified by Fig. 3 showing the 90% percentile response times for the 11 operations (cf. Table 1) specified in XMach-1. Operations Q1-Q8 are queries ranging from document retrieval (Q1) to complex queries involving join, sort and aggregation (Q7, Q8). M1-M3 are data manipulation operations including document insert (M1), document deletion (M2) and updates (M3). A detailed description of the operations can be found in [1]. The measurements for this and the following experiments were carried out on an Intel Pentium III computer running at 800 MHz having 512 MB of main memory and a 40 GB IDE hard disk. The database size was 1000 documents.



	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	M1	M2	M3
old version	250	2684	120	80	30	2713	6709	370	1E+05	32306	291
new version	40	81	10	40	10	50	10	10	671	5518	570

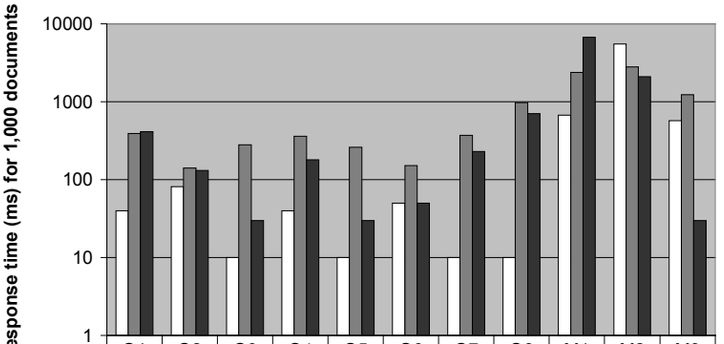
Fig. 3. Comparison of 90th percentile response times for old and new version of a native XML database (NXD 1) in single-user mode

Table 3. Changes in database size for a native XML database system (NXD 2)

	data (MB)	index (MB)
NXD 2 version 1	64,4	72,4
NXD 2 version 2	44,5	35,6
NXD 2 version 3	25,9	31,9

Fig. 3 indicates substantial performance improvements of up to several orders of magnitude between the two versions. All eight query types could be executed with a single-user response time of under 100 ms, favored by the small database size. Some of these improvements were achieved by an optimized benchmark implementation utilizing features of the new version of the system. The high response times for M1 and M2 in both versions stem from an inefficient implementation of the full-text index which has to be updated.

We also observed significant improvements w.r.t. the space requirements to store and index a certain amount of raw XML data. As an example, Table 3 compares the database sizes for data and indexes of consecutive versions of another native XML database system NXD2. The databases were populated using the 1000 documents configuration of the benchmark which has a raw data size of about 16 MB. The changes in the depicted database system resulted in a reduction of the database size of about a factor of 2.5. In general, a ratio of 1.5 to 3 between the database size and the size of raw data is typical for current native XML database systems. Database population takes between 90 and 600 seconds with the 1000 documents setting. Including index generation, 200 to 800 seconds are needed primarily due to full-text indexing. This corresponds to a poor loading throughput of about 20 – 80 KB raw data per second which would result into unacceptable loading times for larger databases.



	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	M1	M2	M3
NXD 1	40	81	10	40	10	50	10	10	671	5518	570
NXD 3	391	141	280	361	260	151	371	971	2374	2804	1231
RDB	411	131	30	180	30	50	230	701	6740	2093	30

Fig. 4. Comparison of 90th percentile response times in single-user mode

When we recently started our evaluation of XML-enabled relational database systems (XRDB) we still found functional shortcomings compared to native XML database systems. One major drawback is insufficient support of an XML query language.

Whereas native XML databases have at least a complete XPath implementation and are starting to support XQuery as well, XRDB's have in most cases only a limited support of XPath with restricted utilization of indices. Another problem is that current XRDBs cannot efficiently run queries across multiple schemas because of their schema driven XML architecture. On the other hand, XRDBs benefit from mature relational functionality and comparably efficient full-text support.

In Fig. 4 and Fig. 5 we compare results for current XML databases running XMach-1. Systems NXD 1 and NXD 3 are commercial native XML databases whereas system RDB is a standard relational database system. The relational implementation of XMach-1 uses a newly developed middleware for a generic mapping of XML data to relations. The RDB mapping is independent of an XML schema and uses only three tables for elements, attributes and large element contents. The mapping supports a sophisticated numbering scheme for XML nodes incurring low renumbering effort for updates as well as a fast navigation and extraction of document fragments.

The 90th percentile response time of all XMach-1 operations for the three database systems are shown in Fig. 4. As can be seen NXD 1 outperforms both other systems in most query operations by an order of magnitude. However update operations are quite slow which stems partly from the full-text index deficiency. The fast execution of Q7 and Q8 was achieved by using extra data structures automatically maintained by database triggers. These optimizations for queries come at the expense of increased overhead for loading and inserting/deleting the XML documents. NXD 3 and RDB exhibit comparable performance figures. The mentioned mapping approach with its optimizations was key to the remarkably good query performance of RDB.

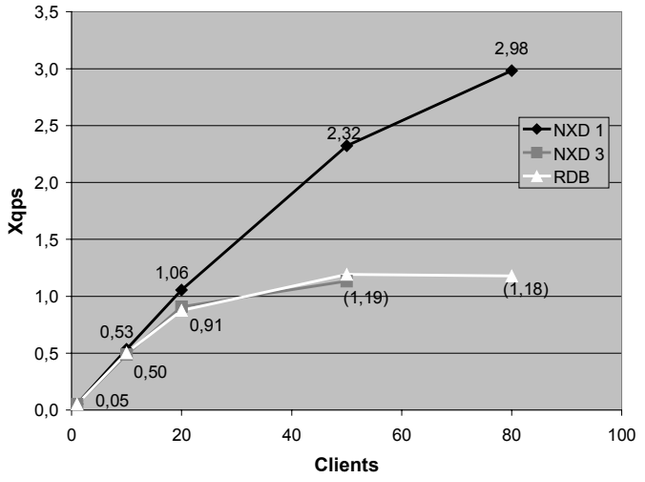


Fig. 5. Throughput comparison

Fig. 5 illustrates the multi-user performance of the three systems. The throughput value Xqps (XML queries per second) measures the number of Q1 operations per second within the query mix. Each client has to wait between two consecutive re-

quests for 1-10 seconds. The small database size largely excludes congestion accessing the external storage devices. Therefore throughput is bound by CPU and locking bottlenecks raised by update operations. NXD 1 again reaches the best value and scales nearly linearly until 50-80 clients. With approximately 3 Xqps it achieves the top value of all evaluated XML database systems so far. Both other systems reach their maximum with 0.9 Xqps and 20 clients. For more clients somewhat higher throughput values (shown in parentheses) were achieved but without meeting the 3 second response time limit of XMach-1.

We started to examine multi-user performance for larger databases requiring a higher degree of IO activity. Some systems had significant scalability problems preventing the execution of some query types. Response times for some queries were order of magnitudes higher than with a 1,000 documents collection. Our XML-to-relational mapping also faced scalability problems for larger data sizes since the relational query optimizer cannot use information provided by the numbering scheme.

6 Conclusion

XML database benchmarks allow to compare the performance of XML database systems for a well-defined environment and operation mix. We reviewed and compared four proposed benchmarks to reveal their primary focus and applicability. From these benchmarks, XMach-1 is the only one supporting the performance evaluation for both single-user and multi-user processing. Moreover it considers not only the query processor but measures the performance of an entire XML database system. Furthermore, it has a focus on document-centric XML databases and considers schema-less and schema-based document collections.

Our experiences with XMach-1 have shown that functionality and performance of XML database systems have considerably improved during the last two years. Native systems have generally performed better than XML-enabled relational database systems. Still, our prototype implementation of a generic XML to relational mapping indicates that generic XML data management on relational databases can reach comparable performance to native XML databases at least for smaller databases. Most XML database systems still face significant scalability problems with larger data volumes and multi-user mode. Hence, there is a big need for further performance improvements and enhanced implementation concepts for XML data management.

References

[1] Böhme, T.; Rahm, E.: XMach-1: A Benchmark for XML Data Management. In Proceedings of German database conference BTW2001, pp. 264-273, Springer, Berlin, March 2001.

[2] Florescu, D.; Kossmann, D.: Storing and Querying XML Data using an RDMBS. In: IEEE Data Engineering Bulletin, Volume 22, Number 3, pp. 27-34, September 1999.

[3] Florescu, D.; Kossmann, D.; Manolescu, I.: Integrating Keyword Search into XML Query Processing. In: Proc. of the 9th WWW Conference, Amsterdam, June 2000.
[4] Böhme, T.; Rahm, E.: Benchmarking XML Data Management Systems. http://dbs.uni-leipzig.de/en/projekte/XML/XmlBenchmarking.html, June 2002.
[5] Schmidt, A.; Waas, F.; Kersten, M. L.; Florescu, D.; Manolescu, I.; Carey, M. J.; Busse, R.: The XML Benchmark Project. Technical Report INS-R0103, CWI, Amsterdam, Niederlande, April 2001.
[6] Schmidt, A.; Waas, F.; Kersten, M. L.; Florescu, D.; Carey, M. J.; Manolescu, I.; Busse, R.: Why And How To Benchmark XML Databases. SIGMOD Record, Volume 30, Number 3, pp. 27-32, September 2001.
[7] Schmidt, A.; Waas, F.; Kersten, M. L.; Carey, M. J.; Manolescu, I., Busse, R.: XMark: A Benchmark for XML Data Management. In Proceedings of the 28th VLDB Conference, Hong Kong, 2002.
[8] The XOO7 Benchmark. http://www.comp.nus.edu.sg/~ebh/XOO7.html, 2002
[9] Carey, M. J.; DeWitt, D. J.; Naughton, J. F.: The OO7 Benchmark. In Proceedings of the ACM SIGMOD International Conference on Management of Data, pp.12-21, June 1993.
[10] Runapongsa, K.; Patel, J. M.; Jagadish, H. V.; Al-Khalifa, S.: The Michigan Benchmark. http://www.eecs.umich.edu/db/mbench/description.html, 2002.
[11] Böhme, T.; Rahm, E.: Benchmarking XML Database Systems – First Experiences. Ninth International Workshop on High Performance Transaction Systems (HPTS), Pacific Grove, California, October 2001.

The Michigan Benchmark: A Microbenchmark for XML Query Processing Systems*

Kanda Runapongsa, Jignesh M. Patel, H. V. Jagadish, and Shurug Al-Khalifa

University of Michigan
Ann Arbor, MI 48109-2122, USA
{krunapon,jignesh,jag,shurug}@eecs.umich.edu

With the continued growing popularity of the eXtensible Markup Language (XML), it is clear that large repositories of XML data will soon emerge. Several methods for managing XML databases have emerged, ranging from retrofitting commercial RDBMSs to building native XML database systems. There has naturally been an interest in benchmarking the performance of these systems, and a number of benchmarks have been proposed [1, 2, 3]. The focus of currently proposed benchmarks is to assess the performance of an XML query processing system when performing a variety of tasks for some representative application. Such benchmarks are valuable to potential users of a database system in providing an indication of the overall performance of the entire application.

One aspect that current XML benchmarks do not focus on is the performance of the basic query evaluation operations such as selections, joins, and aggregations. A "microbenchmark" that highlights the performance of these basic operations can be very helpful to a database developer in understanding and evaluating alternatives for implementing these basic operations. We propose such a microbenchmark, which we call the Michigan benchmark. In this benchmark, we primarily attempt to capture the rich variety of data structures and distribution possible in XML, and to isolate their effects, without imitating any particular application. The benchmark specifies a single data set against which carefully specified queries can be used to evaluate system performance for XML data with various characteristics. We have used the benchmark to test and analyze the performance of different XML query processing systems. The benchmark has revealed key strengths and weakness of these systems, and also points to parts in the individual systems that can be further improved.

The most up-to-date information on the Michigan benchmark, and results from using the benchmark, can be found at:
http://www.eecs.umich.edu/db/mbench.

* This work was supported in part by the National Science Foundation under NSF grant IIS-0208852, by an IBM CAS Fellowship, and by a gift donation from IBM.

S. Bressan et al. (Eds.): EEXTT and DIWeb 2002, LNCS 2590, pp. 160–161, 2003.

References

[1] T. Böhme and E. Rahm. XMach-1: A Benchmark for XML Data Management. In *BTW*, Oldenburg, Germany, March 2001. 160

[2] S. Bressan and G. Dobbie and Z. Lacroix and M. L. Lee and Y. G. Li and U. Nambiar and B. Wadhwa . XOO7: Applying OO7 Benchmark to XML Query Processing Tools. In *CIKM*, 2001. 160

[3] A. R. Schmidt, F. Wass, M. L. Kersten, D. Florescu, I. Manolescu, M. J. Carey, and R. Busse. The XML Benchmark Project. In *VLDB*, 2002. 160

XBench – A Family of Benchmarks for XML DBMSs

Benjamin B. Yao[1], M. Tamer Özsu[1], and John Keenleyside[2]

[1] University of Waterloo
School of Computer Science, Waterloo, Ontario, Canada N2L 3G1
{bbyao,tozsu}@uwaterloo.ca
[2] IBM Toronto Laboratory
8200 Warden Avenue, Markham, Ontario, Canada
keenley@ca.ibm.com

Abstract. This paper summarizes the XBench family of benchmarks that are under development at the University of Waterloo. The benchmark identifies various classes of XML databases and applications and proposes a set of benchmarks to accommodate these classes.

1 Introduction

There are a number of benchmarks for XML databases that have recently been proposed. These usually assume a single application, and define the database schema and workload accordingly. These benchmarks are very effective when the database deployment corresponds to this application characterization. However, one could argue that no "canonical" application exists, and therefore a family of benchmarks are needed. In this paper we summarize our work in developing such a family of benchmarks.

2 Database Design

We characterize database applications along two dimensions: application characteristics, and document characteristics. Application characteristics indicate whether the database that the application uses is data-centric or text-centric. In data-centric (DC) applications, the database stores data that are captured in XML even though the original data may not be in XML. Examples include e-commerce catalog data or transactional data that is captured as XML. Text-centric (TC) applications manage actual text documents and use a database of native XML documents. Examples include book collections in a digital library, or news article archives.

In terms of document characteristics, we identify two classes: single document[1] (SD) and multiple document (MD). The single document case covers

[1] "Document", in this context, refers to an XML document, not to a document as defined in the previous paragraph.

S. Bressan et al. (Eds.): EEXTT and DIWeb 2002, LNCS 2590, pp. 162–164, 2003.

	SD	MD
TC	Online dictionaries	News corpus, Digital libraries
DC	E-commerce catalogs	Transactional data

Fig. 1. Classes of XML databases and applications

those databases, such as an e-commerce catalog, that consists of a single document with complex structures (deep nested elements), while the multiple document case covers those databases that contain a set of XML documents, such as an archive of news documents or transactional data. The result is a requirement for a database generator that can handle four cases: DC/SD, DC/MD, TC/SD, and TC/MD (Figure 1).

For the TC/SD and TC/MD classes, we have analyzed a number of databases to statistically characterize them and generalized these to define database schemas for that particular class. In particular, for TC/SD, we analyzed Oxford English Dictionary (OED) [3] and GCIDE [2].

For DC classes, there are not sufficient number of large XML datasets to perform similar analysis. Therefore, our design for TC classes uses TPC-W [4] benchmark. For TC/SD, we "simulate" an e-commerce catalog by defining a database based on the ITEM table along with the AUTHOR, ADDRESS and COUNTRY tables. These are enhanced by two additional tables that do not exist in TPC-W: AUTHOR_2, which includes additional author information such as mailing address, phone, and email, and PUBLISHER, which consists of publisher name, fax, phone, and email address.

DC/MD class consists of transactional data. Therefore, we use the eight basic tables of the TPC-W database and map them to XML documents. In particular, we shred the ORDERS, ORDER_LINE, and CC_XACT tables to create a large number of XML documents.

For actual data generation, we use ToxGene [1], which is a template-based tool facilitating the generation of synthetic XML documents.

3 Workload Design

In keeping with the benchmark design philosophy, we have specified one set of workload for each type of application identified in the previous section. Each set focuses on the particular features of XML documents in that category. Furthermore, we have designed a set of core queries that test a set of core functionality even if the specific formulation of the query may differ for each class of database.

The workload for a given class of database is designed such that, together with the core set, the benchmark covers the use cases that have been specified for XQuery. It is conceivable that at this time some DBMSs may not be able to process all of these queries. However, we expect that most DBMSs will provide full XQuery support in the near future.

References

[1] D. Barbosa, A. Mendelzon, J. Keenleyside, and K. Lyons. ToXGene: An extensible template-based data generator for XML, In *Proceedings of 5th International WebDB Workshop*, pages 49–54, 2002. 163

[2] *GCIDE_XML: The GNU version of The Collaborative International Dictionary of English, presented in the Extensible Markup Language*, Available at http://www.ibiblio.org/webster/, 2002. 163

[3] *Oxford English Dictionary*, Oxford University Press, 1994, Available at http://www.oed.com. 163

[4] Transaction Processing Council. *TPC Benchmark W Specification, Version 1.8*, February 2002, Available at http://www.tpc.org/tpcw/. 163

Data Integration Using Web Services

Mark Hansen[1], Stuart Madnick[2], and Michael Siegel[2]

[1] MIT Sloan School of Management
E53-321, 30 Wadsworth St, Cambridge, MA 021239
khookguy@yahoo.com
[2] MIT Sloan School of Management
E53-321, 30 Wadsworth St, Cambridge, MA 021239
{smadnick,msiegel}@mit.edu

Abstract. In this paper we examine the opportunities for data integration in the context of the emerging Web Services systems development paradigm. The paper introduces the programming standards associated with Web Services and provides an example of how Web Services can be used to unlock heterogeneous business systems to extract and integrate business data. We provide an introduction to the problems and research issues encountered when applying Web Services to data integration. We provide a formal definition of aggregation (as a type of data integration) and discuss the impact of Web Services on aggregation. We show that Web Services will make the development of systems for aggregation both faster and less expensive to develop. A system architecture for Web Services based aggregation is presented that is representative of products available from software vendors today. Finally, we highlight some of the challenges facing Web Services that are not currently being addressed by standards bodies or software vendors. These include context mediation, trusted intermediaries, quality and source selection, licensing and payment mechanisms, and systems development tools. We suggest some research directions for each of these challenges.

1 Introduction

By providing interface standards, Web Services can be viewed as programming paradigm for extracting and integrating data from heterogeneous information systems. It offers significant advantages over currently available methods and tools. These advantages have been widely discussed in the popular Information Technology press[1]. Because the Web Services paradigm is based on a new set of standards (e.g., XML, SOAP, WSDL, UDDI)[2] it promises to enable the aggregation of multiple data sources

[1] "Vendors Rally Behind Web Services Spec", *InformationWeek*, November 27, 2000; "Web Services Move One Small Step Closer To Reality", *InformationWeek*, February 12, 2001.
[2] Section 4 defines these acronyms.

S. Bressan et al. (Eds.): EEXTT and DIWeb 2002, LNCS 2590, pp. 165-182, 2003.

once these standards are supported by the information systems underlying each business process. These standards are being widely adopted in industry as evidenced by Microsoft's .NET initiative and Sun's Java APIs for XML (JAX) extensions to the Java 2 Platform, Enterprise Edition (J2EE). [12]

We believe that, from a research standpoint, it is useful to view Web Services as a paradigm for aggregation. Using that analogy, we investigate the challenges researchers have uncovered related to aggregation. [1][2][3][7][13] and apply these to Web Services. Foremost among these challenges are the issues of semantics and context mediation.

This paper begins with an example illustrating the power of Web Services as a data integration approach in a telecommunications company. It goes on to illustrate how such an application of Web Services is really a form of aggregation. We provide a working definition of aggregation and examine the application of existing aggregation research to Web Services.

We then briefly explore industry support for Web Services and the technology architecture being adopted by most software vendors for applying Web Services to data integration problems. Lastly, we identify some potential challenges facing Web Services, propose additional infrastructure that will be necessary, and point to some promising research that may be applied to create that infrastructure.

2 Example of a Data Integration Architecture Based on Web Services[3]

International Communications (IC) is a worldwide provider of voice and data (Internet) communications services to global corporations. IC has grown by acquisition and has a variety of information systems in different parts of the world that need to be integrated to provide a global view of available services to their global enterprise customers

For example, consider the Global Provisioning System (GPS) required by the corporate headquarters. When a global customer, such as Worldwide Widgets (WW), asks IC to bid on a contract to provide services, IC must turn to its various global subsidiaries to provision the circuits to fulfill this order. The process starts by creating a master order in the corporate GPS. Being able to create a master order implies that the provisioning data from all subsidiaries has been aggregated together into a master data source. It also requires integration with subsidiary support systems (e.g., Trouble Tickets, Usage Statistics). It is an example of intra-organizational aggregation (i.e., aggregating data within an organization).

Once completed, the master order is communicated to each subsidiary to derive the local provisioning plan in their geography.

[3] Although the details are fictitious, this example is based on real examples of Aggregation challenges faced in the telecommunications industry.

2.1 Potential Solutions

IC considered a spectrum of alternatives for building an Aggregator for the GPS, summarized in the table below.

Integration Alternative	Description
Single System	This approach involves replacing all the divisional provisioning components with a single, integrated, system.
Component Interfaces	This approach involves modifying all the divisional components to provide a Web Services interface.
Web Process Wrappers	This approach involves wrapping the existing divisional components with a thin layer of code to provide a Web Service interface.

IC wanted to implement the Single System alternative because it would standardize meta-data throughout the organization and reduce the amount of custom code development and maintenance required to aggregate divisional data up to corporate. However, there were several problems that prevented IC from pursuing this option. First, replacing all the divisional systems would be a multi-year, hugely expensive, project that would require complete retraining the existing divisional Information Technology (IT) employees and end users. Expensive consultants would be needed to assist with installation, configuration, and extensive retraining.[4] Additionally, IC was acquiring companies and needed a quick way to integrate them with corporate systems.

Considering these challenges, IC decided to implement a five-year plan to standardize divisional systems. In the mean time, IC decided to create custom interfaces between divisional and corporate systems. By building prototype Web Services interfaces for one division, IC determined that this approach leveraged local knowledge to quickly create useful interfaces to the GPS. Some divisional systems had interfaces where the fast and simple task of building Web Process Wrappers was sufficient. In other cases, more work was required to modify a divisional system to create a Component Interface supplying Web Services to the GPS.

2.2 Implementing Web Services Interfaces

Implementing the integration architecture using the Web Services paradigm implied using the following standards for systems integration (See Section 4 for more discussion of these standards.):

---

[4] Lisa Vaas, "Keeping Air Force Flying High," *eWeek*, 22 October 2001, available at http://www.eweek.com/print_article/0,3668,a%253D16944,00.asp / Excerpt: "…The outcome wasn't good. After three painstaking years and a substantial investment — Dittmer declined to quote a cost — a mere 27 percent of the original code's functionality had been reproduced. Originally, Dittmer said, they had expected to retrieve 60 percent of functionality. Eventually, the Air Force killed the project. … Rewriting the systems from scratch would have eaten up an impermissibly large chunk of the Air Force's budget. 'We don't have the money to go out and say, 'OK, let's wholesale replace everything,' Jones said …"

- Data would be communicated between systems in a standard XML format.
- SOAP would be used to send and receive XML documents.
- Aggregation interfaces specifications would be defined with WSDL.
- A registry of all system interfaces would be published using the UDDI.

The Web Services interfaces between the Global Provisioning System and the systems in "Division A" are illustrated below (Fig. 1) such as Provisioning, Trouble Tickets, and Usage Statistics. Similar interfaces would be needed for all the divisions.

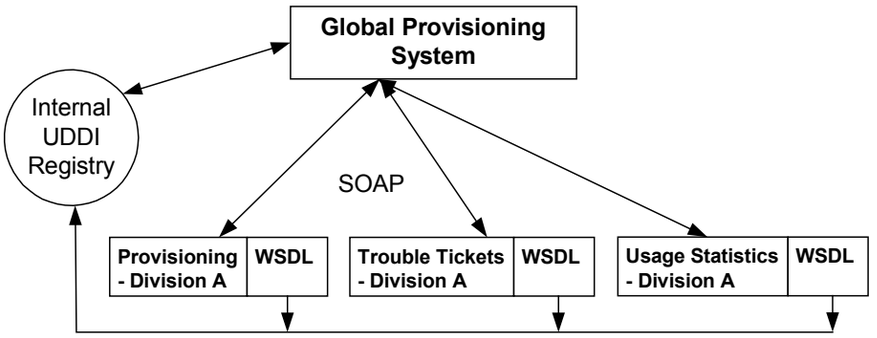


Fig. 1. International Communications' Web Services Interfaces

3 Web Services and Aggregation

Research on information aggregation has been going on for a long time, but with the advent of the Internet there has been a new focus on the entities that aggregate information from heterogeneous web sites – often referred to as "Aggregators"[1]. Much of this research focuses on the semantic and contextual challenges of aggregation [5][7], and as we will see in Section 7 many of these challenges remain in the Web Services paradigm.

Web Services do, however, solve a number of the technical challenges faced by early Internet Aggregators. These Aggregators had to overcome technical challenges related to integration of data source sites that were not originally developed with the intent of supporting aggregation. Screen scraping and "web farming" [4] techniques were developed where the Aggregator accessed the source site as if it were a user and parsed the resulting Hyper Text Markup Language (HTML) to extract the information being aggregated.

The Web Services paradigm solves some of the technical integration challenges by standardizing the infrastructure for data exchange. However, the Web Services paradigm also assumes that application components are designed with the intention of being aggregated. This assumption, that disparate data sources are going to be designed and implemented with the intention of being aggregated, raises a whole new set of challenges that we discuss in Section 7.

To begin exploring the challenges posed by the Web Services paradigm for aggregation, we propose the following definition that encompasses both information and processes aggregation.

An Aggregator is an entity that:

- Transparently collects and analyzes information from different data sources;
- Resolves the semantic and contextual differences in the information;
- Addresses one or more of the following aggregation purposes / capabilities:
  - Content Aggregation
  - Comparison Aggregation
  - Relationship Aggregation
  - Process Aggregation

3.1 Aggregation Purposes / Capabilities

From this definition, we see that not every system designed to integrate data can be called an Aggregator. To be an Aggregator, a system must provide certain capabilities, as summarized here.

Aggregation Capability	**Definition**	**Example**
Content Aggregation	Pulls together information related to a specific topic (e.g., IBM Corporation) and provides value-added analytics based on relationships across multiple data sources.	Employee Benefits Portals where an employee can get access to all his benefits information (e.g., health plan, 401K, etc.)
Comparison Aggregation	Within a particular business domain identifies the optimal transaction based on criteria supplied by the user (e.g., price, time).	Shopbots that compare product prices (e.g., www.mysimon.com, www.dealtime.com).
Relationship Aggregation	Provides a single point of contact between a user and several business services / information sources with which the user has a business relationship.	Aggregation of all your frequent flyer programs (e.g., www.maxmiles.com) or financial accounts (e.g., www.yodlee.com).
Process Aggregation	Provides a single point of contact for managing a business process that requires coordination across a variety of services / information sources.	B2B and EAI tools that provide rule-based workflow and data aggregation to link multiple business processes together (e.g., WebMethods, BizTalk)

3.2 Aggregation Setting

Aggregation types get applied in different settings and have more or less relevance depending on the setting. Three common settings where aggregation is employed are:

- Intra-Organizational – to integrate systems and data within an organization. Process Aggregation is particularly important here where it is often referred to as Enterprise Application Integration (EAI).
- Inter-Organizational – to integrate systems and data across multiple organizations. All aggregation capabilities are important in this context. Process Aggregation is used in many forms of Business-to-Business (B2B) communication such as Supply Chain Management. Many of the Business to Consumer (B2C) Aggregators employ Content Management (e.g., MyYahoo[5]), Comparison (e.g., MySimon[6]), and Relationship (e.g., Yodlee[7]) capabilities.
- Market/Exchange – to create an independent organization and systems to facilitate commerce among members. Process Aggregation, Content Management, and Comparison capabilities are particularly important in this context. A good example is The World Chemical Exchange (www.chemconnect.com) where you can solicit bids from vendors (Comparison), browse and learn about trading partners (Content Management) and buy, sell, and integrate your supply chain with other vendors (Process Aggregation).

The International Communications example represents an Intra-Organizational setting.

4 Web Services Standards – Current State

The Web Services paradigm provides a new set of standards and technologies that facilitate an organization's ability to integrate data from internal heterogeneous systems (e.g., Enterprise Application Integration (EAI)) or integrate data from business partners (e.g., Supply Chain Management and other Business-to-Business (B2B) type applications). These types of systems can be characterized as various types of Aggregators.

For our purposes, we define a Web Service as an application interface that conforms to specific standards in order to enable other applications to communicate with it through that interface regardless of programming language, hardware platform, or operating system. A Web Service interface complies with the following standards:

- XML (eXtensible Markup Language[8]) documents are used for data input and output.

[5] See www.my.yahoo.com
[6] See www.mysimon.com
[7] See www.yodlee.com
[8] See www.w3.org/XML

- HTTP (Hypertext Transfer Protocol[9]) or a Message Oriented Middleware (MOM) product (e.g., IBM's MQ Series) is the application protocol.
- SOAP (Simple Object Access Protocol[10]) is the standard specifying how XML documents are exchanged over HTTP or MOM.
- WSDL (Web Services Description Language[11]) is used to provide a meta-data description of the input and output parameters for the interface.
- UDDI (Universal Description, Discovery and Integration[12]) is used to register the Web Service.

Although there is no single standard for XML document structure, many Web Services that are designed to work together will standardize on a particular set of tags or document structure. Various industry groups and standards bodies are publishing XML standards for use in particular contexts. One example that is building support among technology vendors is ebXML (www.ebxml.org).

4.1 How Standards are Used for Aggregation

Figure 2 illustrates a generic example of how Web Services standards are employed for Aggregation. This is a generic version of Fig. 1 where the box labeled "Aggregator" replaces IC's Global Provisioning System. The programmers developing this system need to integrate the provisioning data provided by systems in various divisions. They accomplish this task by defining standard XML document types as needed (e.g., Order, Provisioning). These documents make use of standard tags for data such as price and bandwidth.

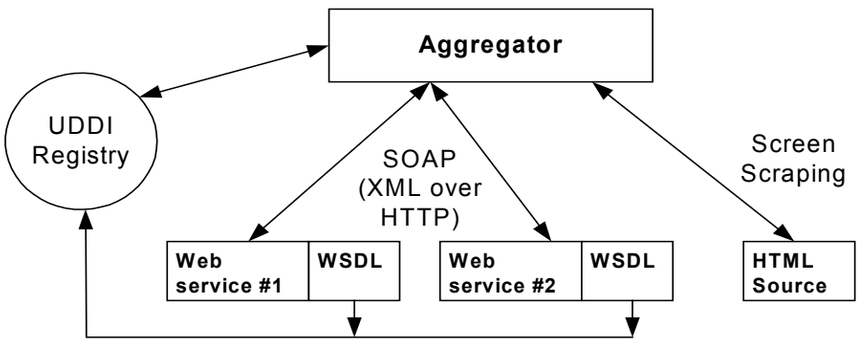


Fig. 2. Aggregation with Web Services

[9] See www.w3.org/Protocols
[10] See www.w3.org/2000/xp
[11] See www.w3.org/TR/wsdl
[12] See www.uddi.org

Within each division, programmers develop a Web Service that can receive and process a query about the network provisioning available (e.g., what bandwidth frame relay connections are available between points A and B?). The interface for each division's Web Service is published using WSDL and registered in a UDDI Registry. The programmers working on the Global Provisioning System can use the UDDI Registry to look up the Web Services that the divisions have made available. From there, they can access the WSDL for each web service that specifies its inputs and outputs.

Some of the divisional Provisioning Systems may be simple enough that instead of implementing a Web Service interface, basic screen scraping off an existing HTML interface is used.

5 Aggregator Architecture

An Aggregator combines data from a variety of sources to create and maintain a new data source supporting new business processes. A standard technical architecture is emerging for creating Aggregators, and is illustrated in Figure 3. Many commercial products are based on such an architecture.

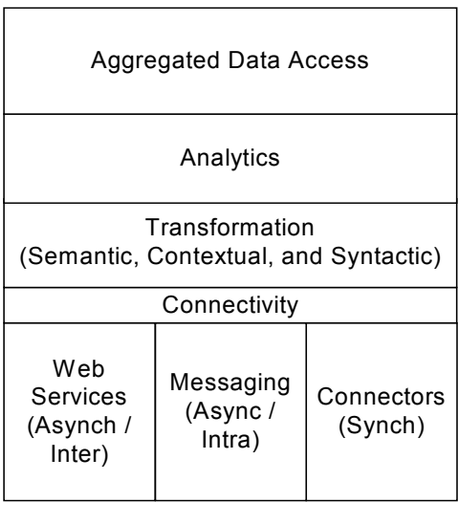


Fig. 3. Aggregation Platform

The Reporting and GUI Access components of this architecture enables the aggregated data to be treated as a single data source and provides tools for querying it as such (e.g., SQL). The Event Handling and Workflow functionality provided by

such platforms provides Process Aggregation that is referred to as Enterprise Application Integration (EAI) if it involves data sources (as in our IC example) or B2B integration if it involves data from different companies (e.g., supply chain integration). All the components below this are designed to leverage Web Services standards for data aggregation.

The Global Provisioning System would use a system architecture like that illustrated in Figure 3. When IC needs to provision a global order, the order is translated into an XML document that represents a query against the "Aggregated Data Access" layer – a virtual or physical (e.g., data warehouse) aggregation of all provisioning data. The resulting provisioning plan is passed down to each local system to create a local image of the provisioning plan for fulfilling the order in the local geography.

In this scenario, the Aggregation Platform builds an aggregated image of the underlying data sources that can be accessed and queried through the "Aggregated Data Access" layer. Other layers in the technology stack perform the following functions.

The Analytics component assembles divisional provisioning plans into a coherent whole – removing data redundancy, resolving conflicts, and optimizing the resulting network structure.

The Transformation component handles standardizing the context and semantics of the information contained in the XML provisioning documents received from local systems. For example, one system may represent bandwidth in bits per second, while another may use megabits per second. This transformation process is one component of business process aggregation that has not been standardized within the Web Services paradigm and is often one of the most difficult integration challenges to overcome.

For example, IC has no standard customer number for WW. Each local system that has been providing network services to local divisions of WW has their own customer number and other information (e.g., address, spelling of name). This is a challenge because the Billing System, for example, needs to aggregate usage data across all of WW and has no standard context (e.g., customer number) for accomplishing that. Often called the Corporate Household or Corporate Family Structure problem [16][17], the issue is that IC has been doing business with local branches and subsidiaries of WC for years under may different names (e.g., Worldwide Consultants, Inc., WC Tokyo Corp., etc.).

We will discuss this problem further in Section 7 when we discuss additional infrastructure that is needed to realize the full potential of Web Services.

The Connectivity Component uses the appropriate method for transmitting data between the divisional components and corporate system. In the IC example, this connectivity is accomplished with the Web Services paradigm using asynchronous exchange of XML documents (perhaps over a corporate message queuing system such as IBM's MQ Series).

5.1 Analytics

The Analytics component extracts data elements from the XML documents exchanged with the Web Services and puts them into a data structure (e.g., relational

database) that can be accessed by the Aggregated Data Access module – perhaps as a type of data warehouse. Analytics also performs analysis that may be useful to decision making that is part of the business process. For example, a bid from one of IC's partners typically contains volume discounts and different pricing for different times of day. The Analytics component will run a model of projected end customer usage of that partner's services to get a projected cost for doing business with that partner. In this manner, it is used by the Event Handling and Workflow module to manage a new business process.

5.2 Transformation

The Transformation component transforms the incoming XML into a standard format with a shared semantics and syntax. For example, if bids come in local currencies, the Transformation component will standardize on U.S. using a pre-determined exchange rate.

5.3 Connectivity

The Connectivity component handles the Web Services function calls using the standards discussed above (e.g., SOAP, XML, WSDL). In addition, an Aggregation architecture would typically provide two other methods for exchanging information with the sources being aggregated. One would be a messaging interface, employing something like IBM's MQ Series for asynchronous communication that is intra-organizational. The other would be a connector interface that provides synchronous connections with intra-organization enterprise computing platforms (e.g., SAP, PeopleSoft). Such connectors may be implemented using standards such as Java's J2EE Connector Architecture.

6 What is New about Aggregation Using Web Services?

Aggregation has been going on long before Web Services standards emerged. What is new is the advent of universally accepted standards for accessing information from heterogeneous sources. These standards will have a profound impact on aggregation and on systems development in general. While early successful Aggregators like Yodlee focused primarily on aggregating data, the next generation of Aggregators will be able to aggregate business processes to create new business models faster and more cost effectively than ever before.

6.1 Standards Make Developing Aggregation Solutions Easier

The standards discussed in Section 4 make aggregation easier because they provide programmers with a common set of productivity tools to work with. Such tools allow developers to spend less time resolving data syntax issues and more time on semantic challenges. For example:

- Fast, easy to use XML parsers are available (e.g., Apache's Xerces and Xalan[13]) for a wide range of programming languages.
- HTTP has become a nearly universally available transport protocol. Where higher fault tolerance is required, standards, such as Java Message Service[14] (JMS), are now available as a common interface to most MOM products.
- SOAP eliminates the need for developers to learn and work with vendors' proprietary data transport protocols. As it matures, most commercial MOM and data integration products are supporting SOAP. SOAP's primary drawback is that, as a text-based protocol, it requires higher bandwidth than the proprietary binary protocols used in many vendor solutions. This issue comes up primarily in high volume, transaction oriented messaging systems and is not as important for typical data aggregation solutions.
- WSDL provides a standard form of documentation to developers who need to write code accessing multiple web services. This reduces the learning curve usually associated when dealing with APIs or other kinds of interfaces to multiple systems.

Lastly, a standard architecture, as show in Figure 3, enables the aggregation problem to be broken down into component parts that can then be "plugged in" to the architecture. For example, one could develop an analytics engine specifically designed for solving semantic issues in financial data aggregation and plug it in to one of the commercial products designed around this architecture. Again, this frees up developers to focus energy on the semantic challenges of financial information aggregation rather than the systems integration challenges of building a custom data aggregation architecture.

6.2 Comparison with Electronic Data Interchange (EDI)

What have enabled this change are the ubiquity of the Internet and the standardization of syntax (XML) and protocols for exchanging information. As a forerunner to Web Services, EDI provided standard protocols and syntax, but required the installation and maintenance of a network linking buyers and suppliers. Today, nearly all businesses have Internet access, and Web Services standards promise to enable much richer business-to-business interaction than EDI.

6.3 Implications of Web Services for Aggregation

There are some key differences between Aggregation via Web Services and traditional approaches:

- *Ease of use* – a great deal of early aggregation (e.g., www.maxmiles.com for aggregating frequent flyer information) was accomplished through screen scraping of web sites that were not designed to be aggregated. Since individual web services will design themselves to be aggregated, process aggregation should be much easier.

[13] See http://xml.apache.org
[14] See http://java.sun.com/products/jms/

- *Standards Based* – aggregation will be facilitated by the acceptance of standards for the exchange of information (e.g., SOAP, XML, WSDL) unlike the early data Aggregators that relied primarily on screen scraping approaches. Also syntax standards like ebXML will reduce custom coding for translation.
- *Products* – early aggregation systems were custom developed by the Aggregators. Because aggregation is now recognized as a potentially large market (e.g., Supply Chain Integration), major software vendors are releasing products that are specifically designed to support aggregation/integration via Web Services. Microsoft's BizTalk[15], IBM's WebSphere Business Integrator[16], and BEA Systems' WebLogic Integrator[17] are examples of such products.

6.4 Cooperative Business Models

Process aggregation will focus less on disinter mediation (e.g., MaxMiles) and more on cooperative models for working with the aggregated enterprises. This is a direct result of the need for permission to access an aggregated enterprise's Web Services in most cases. Hence, like in the IC example given above, we are most likely to see process aggregation used in areas like Supply Chain Management, where multiple organizations need to coordinate business processes.

In addition, we are likely to see a number of "open" free Web Services that can be accessed by anyone over the Internet. Although it is not clear what business model would support such services, a number already exist. Some of these are curiosities (e.g., a prime number tester at http://spheon-jsoap.sourceforge.net/ webservices.php #isPrimeNumber). Others have more real business benefit (e.g., a credit card authorization service at http://sal006.salnetwork.com:83/userman/ ccard/ccard.htm). See www.xmethods.com for a list of free Web Services.

Finally, we are seeing that major software vendors like SAP are beginning to offer Web Services interfaces to their products free of charge. In most cases, these Web Services are simple wrappers around the product's existing APIs.

6.5 Better Return on Investment (ROI)

Process aggregation will improve the ROI of systems integration, business process re-engineering, and B2B applications development. That is because the Web Services standards and the vendor products supporting them via the aggregation architecture illustrated above will make it much faster and easier to aggregate business processes. Many of theses complex integration tasks can now be reduced to defining XML interfaces between an aggregator and an aggregated system. In addition, reusability will be improved, because once a Web Services interface is developed, it can be used by multiple integrators.

---

[15] www.microsoft.com/biztalk/default.asp
[16] http://www-4.ibm.com/software/webservers/btobintegrator/index.html
[17] www.bea.com/products/weblogic/integration

7 Challenges and Potential Research Directions

Today's Web Services standards specify common protocols for the exchange of information between systems. Other efforts, like ebXML (www.ebxml.com), target the standardization of syntax and protocols to standardize common business transactions (e.g., invoicing). However, there are still many significant challenges that remain in order for the Web Services paradigm to meet the integration requirements of many aggregation challenges. These challenges are summarized in the table below and explained in the following sub-sections.

Challenge	Brief Description
Semantics	Different Web Services will have different meanings attached to data values that may have the same, standard, name in each service. The challenge is to mediate between these different contexts.
Modularization of Business Processes	Existing EIS solutions (e.g., SAP) are monolithic and not easy to break into modular pieces of functionality to facilitate "best of breed" computing.
Security and Trusted Intermediaries	What methods will be most effective for ensuring that only authorized users can access a Web Service? Conversely, how does a user ensure that a Web Service does not misuse information that is exchanged during interaction?
Quality and Source Selection	The challenge is to ensure that a Web Service is providing accurate, complete, consistent, and correct information. Given the potential for multiple Web Services providing similar capabilities, how select most appropriate source?
Licensing and Payment Mechanisms	How will users pay for access to Web Services?
Development Tools	What kind of tools (e.g., modeling, programming, search) will be needed to make Web Services development efficient?

7.1 Semantics

Web Services Description Language (WSDL) is used to specify the XML syntax required to communicate with a Web Service. However, problems can still arise related to inconsistent meanings, or semantics.

Consider, for example, a Web Service provided by each of Global Telecom's divisions to return bandwidth data when queried about a particular customer's network connection between two points. One division's Web Service may represent bandwidth in bits per second, while another may use megabits per second. This transformation process has not been standardized within the Web Services paradigm and is often one of the most difficult integration challenges to overcome.

The bandwidth problem can be solved by defining a new type, called "mbs" for "megabits per second," and then using this type for the variable Bandwidth.

Assuming that the programmers writing this Web Service in each division implement the WSDL specification correctly, then each would convert their units for bandwidth into megabits per second.

Some semantic problems, like the bandwidth units, can be overcome by specifying unique types. However, this is not always possible or practical. Consider a Web Service provided by each division that requires a "customer number" to retrieve local usage information for corporate billing purposes.

Commonly, organizations like IC do not have standard customer numbers for their clients. For example, each local system that has been providing network services to local divisions of WC probably has its own customer number and other information (e.g., address, spelling of name). This is a challenge because the Billing System, for example, needs to aggregate usage data across all of WC and has no standard context (e.g., customer number) for accomplishing that. Often called the Corporate Household or Corporate Family Structure problem[16][17], the issue is that IC has been doing business with local branches and subsidiaries of WW for years using a variety of customer numbers. Importantly, even XML schema standardization efforts like ebXML do not solve this Corporate Household problem.

7.1.1 Context Mediation

One solution may be to introduce Context Mediation into the Web Services paradigm [6][7]. In the IC example, a Context Mediation Service would identify and resolve potential semantic conflicts between the user and provider of a Web Service.

An example of such a Context Mediation framework is provided by MIT's COntext INterchange (COIN) project [2][7][8][9][10][11]. Following the COIN model, with the Web Services framework there would be standards to supply:

- A Domain Model to define rich types (e.g., customer number).
- Elevation Axioms to apply the Domain Model to each Web Service and define integrity constraints specifying general properties of the Web Service.
- Context Definitions to define the different interpretations of types in each Web Service (e.g., CustomerName might be "division level" or "corporate level").

The W3C is doing similar work in the context of its "Semantic Web" initiatives (www.w3.org/2001/sw/) that could be leveraged to provide standards for this type of Context Mediation. For example, a Domain Model standard could be defined as a subset of XML Schema (www.w3.org/XML/Schema).

Another approach to adapting the COIN model for Context Mediation to Web Services is suggested by the work being done on RuleML [14][15]. RuleML is XML syntax for rule knowledge representation. It is designed to be inter-operable with commercially important families of rule systems such as SQL, Prolog, Production rules, and Event-Condition-Action rules (ECA). For example, the Elevation Axioms used by COIN to mediate different contexts could be stored in RuleML in a Web Service's WSDL, or in a local UDDI directory.

If there were clear standards for these components of Context Mediation, then the vendors providing Aggregation tools, with architectures like that exhibited in Figure 3, could build Context Mediation capabilities into their products just as they have built in support for Web Services standards like SOAP, WSDL, and UDDI.

7.2 Modularization of Business Processes

It may prove very difficult to modularize the business processes, as automated in EIS packages like SAP and Siebel. Apart from the programming challenges related to adding Web Services features to these products, there are ontological challenges to modularization.

For example, at IC, many of the divisions have Order Management Systems that automatically generate a new customer in the local Billing System each time a new order is provisioned. The databases behind these Order Management Systems often enforce referential integrity between orders and the customer database in the Billing System. So, to avoid rewriting a lot of code in order to aggregate these local systems, the Enterprise Order Management System will need to add customer information to each of the local Billing Systems. But this customer information will also reside in the Enterprise Billing System, so we now need to maintain consistency across all these systems, and modify the local Billing System to not bill the local division of WW directly, but to roll-up local usage from WC to the Enterprise Billing System.

7.3 Security and Trusted Intermediaries

Publishers of Web Services on the Internet will need a security mechanism to control that is able to access their services. For example, access to a person's credit history should only be available to those with the legal right to obtain that information.

There are several ways that standards could be created, and infrastructure developed to build security into the Web Services paradigm. One possibility is simple password protection. In order to use a particular Web Service one would have to register and receive a user name and password.

Another possibility is to use Public Key Encryption as the basis for a security standard. In this model, anyone would be able to access a Web Service, but the XML documents returned by the service would be encrypted and only authorized users, with the proper key would be able to de-crypt them.

Ensuring the security of a Web Services user is another important consideration. For example, suppose that a company created a Web Service that provided an artificial intelligence based disease diagnosis. For a fee, a customer (or the information systems at a customer's hospital) could supply medical history and symptoms and receive back diagnostic information. Doctors to confirm diagnoses, insurance companies to validate treatments prescribed by doctors, and individual patients themselves, might use such a Web Service. To use such a system, a patient's medical history must be supplied to the Web Service. Clearly, the patient would want to ensure the confidentiality of that information, and also ensure that the company providing the Web Service did not even have access to the information provided.

In this scenario, it might make sense for the user of a Web Service to work through a "trusted intermediary" - an entity that could access Web Services on behalf of the customer and ensure that confidential information is not revealed to the operator of the Web Service.

7.4 Quality and "Source Selection"

Another important issue in the development of the Web Services paradigm is information quality. How does a customer know, for example, that a linear equation solving Web Service is providing correct answers?

Solving this problem (i.e., ensuring the accuracy, consistency, completeness, etc. of results obtained from a Web Service) is difficult. One possibility is the emergence of Web Services auditors that give their seal of approval to individual Web Services much the way that Public Accounting firms audit a company's financial results. Along these lines, the W3C has recently announced the creation of a Quality Assurance (QA) Activity (www.w3.org/QA/). Perhaps some of these issues will be addressed in that forum.

7.5 Licensing and Payment Mechanisms

Suppose you were developing a Financial Advisor site. To offer a complete set of services to customers, you might want to access Web Services for things like stock quotes, yield curve calculations, risk-arbitrage models, etc. One payment scenario would involve you signing licensing agreements with each Web Service – perhaps paying a monthly fee.

Another approach could be a "per use" charge, so that you were charged a small amount each time you accessed the Web Service. The market for Web Services would be helped by the existence of a standard "per use" payment services. If both the Web Services and the Financial Advisor aggregator were members, then the charges would be computed and handled automatically. The service would act as an intermediary, providing monthly statements to the aggregator, collecting fees, and sending payments to the Web Services. One commercial platform that has the potential to become such a service is Microsoft Passport[18].

7.6 Development Tools for Aggregation

To build a system using Aggregation and the Web Services paradigm, developers need tools to locate the Web Services they need to aggregate into their application.

To enable this kind of search, first a language is needed to describe the process that a Web Service is needed for. Perhaps the Unified Modeling Language (UML) could be adapted to this purpose to create a Unified Modeling Language for Web Services (UMLWS).

This is another area where knowledge representation efforts such as RuleML could be helpful. For example, the use of a particular Web Service is probably subject to a number of constraints that may or may not make it suitable for a particular task. Going back to our example, suppose that each division of IC has a "minimum order size" expressed in terms of bandwidth or length of contract. These rules could be expressed as RuleML and stored in the WSDL so that a developer could determine whether or not the Order Management System's Web Service at a particular division can be used for a particular order or not.

---

[18] See www.passport.com

Once standards such as UMLWS and RuleML are devised and adopted, then Web Services Search Engines could be developed that take UMLWS and RuleML as input and search a UDDI directory for Web Services that provide the necessary processes.

8 Conclusion

The infrastructure is falling in place to enable great efficiencies of data integration, both internally within an organization (EAI) and externally across organizations (B2B). The ubiquity of the Internet, along with standardization on TCP/IP and HTTP create near universal connectivity. But connectivity is only the first step toward integration. Today, the Web Services paradigm promises to standardize the syntax and protocols used for communication between applications. This is another important step toward facilitating data integration. However, it is important to remember that many challenges lie ahead. A good first step for researchers would be to implement a prototype aggregation system using commercially available software products and the architecture described in this paper. This would provide a concrete demonstration of the degree to which syntax and protocol challenges have been solved.

However, as the problems of syntax and protocols for integration get resolved, we will find ourselves facing the additional challenges of semantics, modularization of business process, security, and other issues discussed in this paper. It will be interesting to see how work that has been done on Context Mediation, the Semantic Web, and other areas can be applied to meet these challenges.

References

[1] Madnick, S (1999). "Metadata Jones and the Tower of Babel: The Challenge of Large-Scale Semantic Heterogeneity", *Proc. IEEE Meta-Data Conf.*, April 1999.

[2] Madnick, S. (2001). "The Misguided Silver Bullet: What XML will and will NOT do to help Information Integration", Proceedings of the Third International Conference on Information Integration and Web-based Applications and Services (IIWAS2001), September 2001.Madnick, S., Siegel, M. Frontini, M., Khemka, S., Chan, S., and Pan, H., "Surviving and Thriving in the New World of Web Aggregators", MIT Sloan Working Paper #4138, October 2000 [CISL #00-07].

[3] Bressan, S., Goh, C., Levina, S., Madnick, S., Shah, A., and Siegel, M., "Context Knowledge Representation and Reasoning in the Context Interchange System", *Applied Intelligence* (13:2), Sept. 2000, pp. 165-179.

[4] Hackathorn, R (1999). *Web Farming for the Data Warehouse*, Morgan Kaufmann Publishers.

[5] Goh, C. (1996). Representing and Reasoning about Semantic Conflicts in Heterogeneous Information Systems, PhD Thesis, MIT, June 1996.

[6] Tomasic A., Raschid L., and Valduriez P., "Scaling Access to Heterogeneous Databases with DISCO," in IEEE Transactions on Knowledge and Data Engineering, 10(5), 1998.

[7] Goh, C., Bressan, S., Madnick, S., and Siegel, M. (1999). "Context Interchange: New Features and Formalisms for the Intelligent Integration of Information," *ACM Transactions on Office Information Systems*, July 1999.

[8] Goh, C., Bressan, S., Levina, S., Madnick, S., Shah, A., and Siegel, M. (2000). "Context Knowledge Representation and Reasoning in the Context of Applied Intelligence," *The International Journal of Artificial Intelligence, Neural Networks, and Complete Problem-Solving Technologies*, Volume 12, Number 2, Sept. 2000, pp. 165-179.

[9] Goh, C., Madnick, S., and Siegel, M. (1994). "Context Interchange: Overcoming the Challenges of Large-Scale Interoperable Database Systems in a Dynamic Environment," *Proceedings of the Third International Conference on Information and Knowledge Management*, pages 337-346, Gaithersburgh MD.

[10] Siegel, M. and Madnick, S. (1991) "Context Interchange: Sharing the Meaning of Data," *SIGMOD RECORD*, Vol. 20, No. 4, December pp. 77-78.

[11] Siegel, M. and Madnick, S. (1991) "A Metadata Approach to Solving Semantic Conflicts," *Proceedings of the 17th International Conference on Very Large Data Bases*, pages 133-145.

[12] Hansen, M., "Changing Terrain: Open middleware standards are redefining EAI and B2B integration", *Intelligent Enterprise*, August 10, 2001.

[13] Moulton, A., Bressan, S., Madnick, S. and Siegel, M., "An Active Conceptual Model for Fixed Income Securities Analysis for Multiple Financial Institutions," *Proc. ER 1998*, pp. 407-420.

[14] Grosof, B. and Labrou, Y., "An Approach to using XML and a Rule-based Content Language with an Agent Communication Language." In Frank Dignum and Mark Greaves, editors, *Issues in Agent Communication*. Springer-Verlag, 2000.

[15] Grosof, B., "Standardizing XML Rules: Preliminary Outline of Invited Talk", *Proceedings of the IJCAI-01 Workshop on E-business and the Intelligent Web*, edited by Alun Preece, August 5, 2001.

[16] Chen, X., Funk, J., Madnick, S., and Wang, R., "Corporate Household Data: Research Directions", *Proceedings of the Americans Conference on Information Systems* (AMCIS, Boston), August 2001 [SWP #4166, CISL WP #01-03, TDQM WP#2001-08].

[17] Madnick, S., Wang, R., Dravis, F., and Chen, X., "Improving the Quality of Corporate Household Data: Current Practices and Research Directions", *Proceedings of the Sixth International Conference on Information Quality* (IQ2001, Cambridge), November 2001, pp. 92-104 [CISL #01-10].

Efficient Cache Answerability for XPath Queries

Pedro José Marrón and Georg Lausen

University of Freiburg, Institute of Computer Science
Georges-Köhler-Allee, 79110 Freiburg, Germany
{pjmarron,lausen}@informatik.uni-freiburg.de

Abstract. The problem of cache answerability has traditionally been studied over conjunctive queries performed on top of a relational database system. However, with the proliferation of semistructured data and, in particular, of XML as the de facto standard for information interchange on the Internet, most of the assumptions and methods used for traditional systems – and cache answerability is no exception – need to be revisited from the point of view of the semistructured data and query model. In this paper, we present a formal framework for the efficient processing of XPath queries over XML documents in a cache environment that is based on the classic rewriting approach. Furthermore, we provide details on the implementation of our formal methods on top of HLCaches, an LDAP-based distributed caching system for XML, and argue that our approach is more efficient than traditional query rewriting algorithms while, at the same time, supporting the full expressive power of XPath queries.

Keywords: Semistructured data, cache answerability, query rewritability, XML, XPath, LDAP

1 Introduction

Cache answerability has been traditionally studied in the realm of conjunctive predicates and queries performed on top of relational database systems [Lev00], but the increasing interest in recent years on the characteristics and capabilities of semistructured models and, in particular, XML [BPSMM00], have lead to the restatement of the cache answerability problem in terms of the semistructured data and query model [CGLV00, KNS99, PV99]. Furthermore, the proliferation of techniques to perform data integration (let it be semistructured or not) on the Internet strive the need for efficient cache mechanisms.

The use of XPath [CD99] and XPath-based models for the querying and processing of semistructured data has changed the focus of the rewriting algorithms from conjunctive predicates to regular path queries [CGLV00], or other query languages specifically designed for a particular semistructured data model [PV99].

Other query caching systems, like [LRO96], [DFJ+96] or [QCR00], do not take into consideration semistructured data, and although interesting in their approach, cannot be used in the context our model can be brought up.

S. Bressan et al. (Eds.): EEXTT and DIWeb 2002, LNCS 2590, pp. 183–200, 2003.

The approach we take in our work, and therefore, the focus of this paper, is on the definition of a very simple, but highly efficient general-purpose formal model that allows us to tackle the problem of cache answerability for XML from a more pragmatic perspective than the one usually taken by traditional papers on the topic. The generality of our model enables its implementation on any XPath-aware caching system, and in order to show its feasibility, we have implemented it as part of HLCaches [ML01, Mar01], a hierarchical LDAP-based caching system for XML.

In our system, the methods and algorithms described throughout this paper serve as the basis for the efficient processing of XPath queries in the distributed caching environment offered by HLCaches, since it allows the definition of partial XPath query evaluation techniques, query preprocessing mechanisms, and parallel processing routines that are crucial for the maintenance of the level of availability and processing capabilities expected from an distributed caching system.

This paper is structured as follows: Section 2 presents a formal description of the XPath query model needed to understand the reformulation of the cache answerability problem detailed in section 3. Section 4 provides an insight in some of the more important implementation issues related to our model, and section 5 concludes this paper.

2 XPath Query Model

As specified in the XPath standard [CD99], the primary purpose of the XPath query language is to address parts of an XML document, usually represented in the form of a tree that contains element, attribute and text nodes.

An XPath Query Q_X is formed by the concatenation of path expressions that perform walk-like operations on the document tree retrieving a set of nodes that conform to the requirements of the query. Each expression is joined with the next by means of the classical Unix path character '/'.

Definition 1 (XPath Query). *An XPath Query Q_X is defined as:*

$$Q_X = /q_0/q_1/\ldots/q_n,$$

where q_i is an XPath subquery defined below, and '/' the XPath subquery separator. □

Definition 2 (XPath Subquery). *An XPath Subquery q_i is a 3-tuple*

$$q_i = (C_i, w_i, C_{i+1}),$$

where:

- *C_i is a set of XML nodes that determine the input context.*
- *w_i is the Path Expression to be applied to each node of the input context (defined below).*

Table 1. Allowed Axis Expressions in XPath

Axis Name	Considered Nodes
ancestor	Any node along the path to the root
ancestor-or-self	Same, but including the current node
attribute	Consider only attribute nodes in the tree
child	Any node directly connected to the current node
descendant	Any node from the subtree rooted at the current node
descendant-or-self	Same, but including the current node
following	Any node with id greater than the current node, excluding its descendents
following-sibling	Any same-level node with id greater than the current node
parent	The direct predecessor of the current node
preceding	Any node with id lower than the current node, excluding its ancestors
preceding-sibling	Any same-level node with id lower than the current node
self	The current node

– *C_{i+1} is a set of XML nodes resulting from the application of the path expression w_i onto the input context C_i. C_{i+1} is also called the output context.*

□

Definition 3 (XPath Path Expression). *A Path Expression w_i is a 3-tuple $w_i = a_i :: e_i[c_i]$, such that:*

– *a_i is an axis along which the navigation of the path expression takes place (see table 1 for a complete list).*
– *e_i is a node expression that tests either the name of the node or its content type.*
– *c_i is a boolean expression of conditional predicates that must be fulfilled by all nodes in the output context.*

□

Example 1. The query $Q_X = /child :: mondial/child :: country[attribute :: car\_code = \text{“}D\text{”}]$ is composed of two subqueries whose combination selects all `country` nodes directly connected to the `mondial` child of the document root, that have an attribute `car_code` with value “`D`”. □

More formally, and using the classic predicate-based approach found in most rewriting papers, the evaluation of a query Q_X, can be defined in terms of the evaluation of its respective subqueries by means of the following predicate:

Definition 4 (XPath Subquery Evaluation). *Given an XPath subquery $q_i = (C_i, w_i, C_{i+1})$, where C_i is the input context, w_i is a path expression,*

and C_{i+1} the evaluation of w_i on C_i (also called the output context), we define its evaluation by means of the eval predicate, as follows:

$$C_{i+1} = eval(C_i, w_i)$$

where the eval predicate is simply an abbreviation of the following expression:

$$eval(C_i, w_i) = \bigcup_{n \in C_i} (evalNode(n, w_i))$$

where evalNode performs the evaluation of w_i over a single input node, returning all other nodes in the document that satisfy w_i. □

Definition 5 (XPath Query Evaluation). *Given an XPath query Q_X of the form $Q_X = /q_0/\ldots/q_n/$, its evaluation is defined in terms of the eval predicate as follows:*

$$Q_X = C_{n+1} = eval(C_n, w_n), \quad where$$
$$C_{i+1} = eval(C_i, w_i), 0 \leq i \leq n$$

The result of the query is simply the last output context from subquery q_n, that in turn, depends on the output context of q_{n-1}, and so on.

As defined in the XPath standard [CD99], C_0 is said to contain only the root of the document tree. □

Given the highly serial characteristics of the XPath query model, the evaluation process for a given XPath query can be easily visualized using the graphical representation of figure 1, where, as an example, the evaluation process of a query consisting of seven subqueries is depicted. The ovals inside each context between two subqueries indicate the individual XML nodes that satisfy the subquery at each point.

It is worth mentioning at this point that the evaluation of a given subquery q_i involves the application of q_i on each one of the individual nodes contained in the previous context, so that it is possible to keep track of which node in context C_i generates what set of nodes from C_{i+1}.

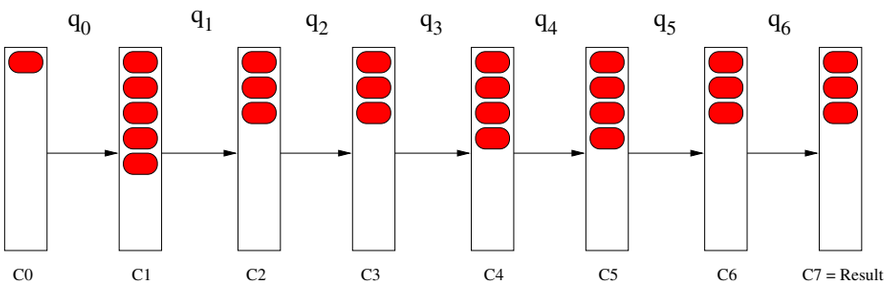


Fig. 1. XPath Query Evaluation Example

3 Cache Answerability

Cache answerability is the basis for more complex problems whose solution usually implies the more efficient processing of queries on a given system. As we have already mentioned in the introduction, data integration, and in particular semistructured data integration is becoming more and more common, and requires efficient cache solutions that must rely on efficient cache answerability algorithms.

The problem of cache answerability is usually reduced to determining, given a particular query, whether or not there exists a rewriting for the query in terms of elements or predicates already known to the cache, assuming that the retrieval of results from the cache can be performed more efficiently than their repeated evaluation. Of course, this is only the case if the collection and maintenance of information in the cache can be implemented in such a way that the path taken by the query for its evaluation is not slowed down by the data gathering phase used as the basis for the query rewriting algorithms.

In our case, in order to fully support the rewriting of XPath queries, we only need to store the output context of each subquery in our cache as it is evaluated in the first place. The predicate $cache(I, w^c)$, defined in exactly the same way the *eval* predicate was introduced in the previous section, is stored in the cache, and contains the set of path expressions with their respective input and output contexts that have been evaluated by the cache thus far. The difference between the *eval* and *cache* predicates lies in the efficiency of their implementation. Whereas the former needs to invoke a parser on the subquery expression and evaluate it on top of the document tree, the latter simply performs a look up on the current contents of the cache to immediately retrieve the answer to the subquery.

Using these definitions, and taking into account that the nature of XPath expressions allows us to perform rewritings at the subquery level, we obtain the following definitions:

Definition 6 (Equivalent Rewriting).
Given an XPath subquery q_i defined by $q_i = (C_i, w_i, C_{i+1})$ that needs to be evaluated by the application of an $eval(C_i, w_i)$ predicate, an equivalent rewriting is another predicate $cache(I^c, w^c)$, such that the following properties hold:

- $w = w^c$; *and*
- $C_i = I^c$.

The evaluation of the subquery q_i is then, $C_{i+1} = cache(I^c, w^c)$. □

However, looking at this definition, it is clear that we can relax the second constraint to allow for a greater number of equivalent rewritings to be detected.

Definition 7 (Weak Equivalent Rewriting).
Given an XPath subquery $q_i = (C_i, w_i, C_{i+1})$ that needs to be evaluated by the application of an $eval(C_i, w_i)$ predicate, a weak equivalent rewriting is another predicate $cache(I^c, w^c)$, such that the following properties hold:

– $w = w^c$; *and*
– $C_i \subseteq I^c$.

The evaluation of the subquery q_i, and therefore, the contents of C_{i+1} is then the set of nodes in the output context generated as a consequence of the evaluation of w_i on the input context C_i, that is, $C_{i+1} = cache(C_i, w^c)$.

In other words, if our cache contains a superset of the answers needed to provide a rewriting for the subquery q_i, we are still able to evaluate the subquery q_i only with the contents of our cache. □

Finally, we can define what it means for a rewriting to be partial:

Definition 8 (Partial Equivalent Rewriting).
Given an XPath subquery $q_i = (C_i, w_i, C_{i+1})$ that needs to be evaluated by the application of a $eval(C_i, w_i)$ predicate, a partial equivalent rewriting is another predicate $cache(I^c, w^c)$, such that the following properties hold:

– $w = w^c$; *and*
– $C_i \supseteq I^c$.

Then, the evaluation of the subquery q_i is the set of nodes in the output context found in the partial equivalent rewriting, plus the set of nodes that needs to be evaluated by means of the eval predicate, that is, $C_{i+1} = cache(I^c, w^c) \cup eval(C_i \setminus I^c, w)$. □

These definitions allow us to create a framework where, independently of the storage model used for XML documents, and taking only the characteristics of the XPath query language into account, the problem of finding equivalent rewritings for a given query and, by extension, the problem of cache answerability can be very easily solved.

Let us illustrate the functionality of our framework with an example:

Example 2. Let us assume that our cache contains an instance of the mondial database [May], where various pieces of information about geopolitical entities is stored. Let us also assume that the query $Q_1 = /mondial/country//city$ has already been evaluated, and its intermediate results stored in our cache by means of several *cache* predicates, namely:

$$\begin{aligned} cache\ contents = {} & cache(C_0, \text{“}/mondial\text{”}), \\ & cache(O_1^c, \text{“}/country\text{”}), \\ & cache(O_2^c, \text{“}//city\text{”}) \end{aligned}$$

where O_1^c and O_2^c are the stored results of the evaluation of “/mondial” on C_0 and “/country” on O_1^c, respectively.

In order to evaluate the query

$$Q_X = /mondial/country[car\_code = \text{“D”}]//city$$

using the *eval* predicate, we need to solve the following expression:

$$\begin{aligned} Q_X = C_3 &= eval(C_2, \text{“}//city\text{”}), \quad \text{where} \\ C_2 &= eval(C_1, \text{“}/country[car\_code = \text{“D”}]\text{”}) \\ C_1 &= eval(C_0, \text{“}/mondial\text{”}) \end{aligned}$$

However, the evaluation of Q_1 provided us with a series of *cache* predicates that we can use in order to rewrite Q_X as follows:

$$\begin{aligned} Q_X = C_3 &= cache(C_2, \text{“}//city\text{”}), \quad \text{where} \\ C_2 &= eval(C_1, \text{“}/country[car\_code = \text{“D”}]\text{”}) \\ C_1 &= cache(C_0, \text{“}/mondial\text{”}) \end{aligned}$$

As we can see from the cache contents detailed above, we can find two equivalent rewritings, one for the first subquery, and another one for the last subquery. The subquery in the middle does not exist in our cache, and therefore, needs to be evaluated by means of the *eval* predicate.

In this example, we can see two different kinds of rewritings: an equivalent rewriting for the */mondial* subquery, since the contents of C_0 are fixed and defined to be the root of the XML data, and a weak equivalent rewriting for the *//city* subquery, since the contents of C_2 are a subset of the contents of O_2^c defined in the cache. This is obvious since the query */country* retrieves all `country` nodes in the document, whereas */country*[*car_code* = “D”] selects only one `country` node from all existing countries. □

4 Implementation Issues

As we have already mentioned, our model has been implemented as part of the query evaluation engine of the HLCaches system, whose basic structure and evaluation algorithms have been published in [ML01]. However, the generality of the model described in the previous section allows for our mechanisms to be implemented and deployed not only in HLCaches, whose XML storage model is based on LDAP [WHK97, HSG99], but on any XPath processing system that follows the XPath standard [CD99].

In order to provide an implementation of our model, the following functionality needs to be provided:

XML Data Model: An efficient storage and retrieval mechanism for XML.

XPath Evaluation Model: The implementation of the *eval* predicate in such a way, that the evaluation of a subquery is completed before the evaluation of the next subquery starts. This requirement is needed due to the highly serial nature of the XPath evaluation model.

Cache Data Model: Storage of cache contents (the *cache* predicate) in structures that allow for their efficient checking and retrieval.

Cache Evaluation Model: Algorithms or query types used to determine the result of a particular *cache* predicate.

In HLCaches, we have implemented this model using the following approaches:

4.1 XML Data Model

LDAP is used in HLCaches as the underlying representation model for the encoding of arbitrary XML documents. The exact representation, as well as the internal details of the storage mechanisms fall out of the scope of this paper, but the interested reader is referred to the aforementioned publications.

For the purposes of our discussion regarding the implementation of our model, it suffices to know that an LDAP server maintains the directory *schema* model (equivalent to a DTD [BPSMM00] or XMLSchema [Fal01, TBMM01] representation), and the directory *data* model.

The directory schema model manages the meta-data about the contents of the LDAP tree, which implies the storage of mainly three types of information:

LDAP Schema Class Hierarchy: Contains information about the **required** and **allowed** attributes a particular class of nodes are able to store, as well as the hierarchical relationships among the different classes of nodes in the tree.

Valid Attributes: Represent the set of recognized attributes as well as their type, which determines the kind of search and modify operations allowed on a specific attribute.

Type Definition: Stores the set of allowable types that can be given to a specific attribute.

The directory instance, on the other hand, manages a set of nodes and their respective attributes whose representation, similarly to XML, is a tree-based structure that can be stored and retrieved very efficiently. The hierarchical structure of an LDAP directory is kept by means of two special purpose attributes: **object class** (or **oc** for short) that stores the set of classes a node belongs to, and a **distinguished name** (or **dn** for short), defined below:

Definition 9 (LDAP Distinguished Name).
An LDAP distinguished name is a comma separated sequence of attribute-value pairs that uniquely identifies a particular node in the LDAP tree.

A distinguished name for a particular entry is formed by taking the distinguished name of the parent node in the hierarchy, and prepending an attribute-value pair unique to all the siblings of the corresponding node. This attribute-value pair is referred to as the ***relative distinguished name****.* □

Since the distinguished name contains each relative distinguished name from a particular node up to the root, and there is only one parent for each node,

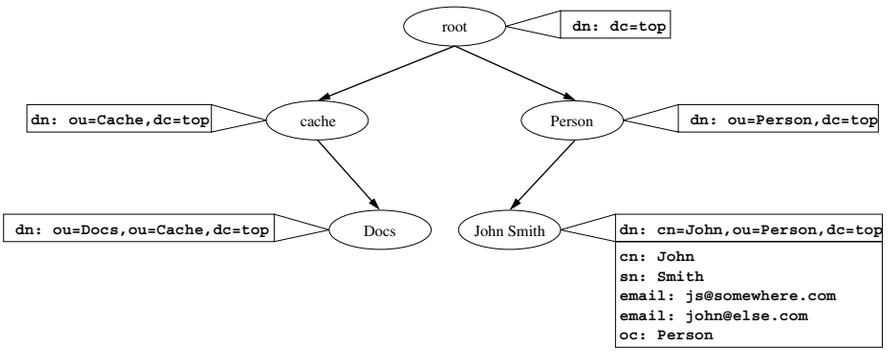


Fig. 2. LDAP Directory Instance Example

the distinguished name is enough to uniquely identify a particular entry in the instance hierarchy.

As it can be seen in the following example, attributes in LDAP are multivalued, that is, there is no restriction on the number of values a particular attribute in a specific node is allowed to take. This allows the **oc** attribute to store all the (potentially many) classes a particular node in the LDAP tree is an instance of.

Example 3 (LDAP Directory Instance). Figure 2 contains a graphical representation of an instance in an LDAP directory, where both, the use of distinguished names to represent the hierarchical relationships, and the purpose of attribute names to store information about a particular node is explicitly stated. □

The similarities between the LDAP and XML model allow us to store XML documents without the need to provide cumbersome transformations like the ones needed to represent XML in relational databases, making LDAP the ideal underlying data model for implementation on a cache.

4.2 XPath Evaluation Model

Similarly, the Lightweight Directory Access Protocol offers a querying model based on filter specification that happens to be very close in nature to that of native XPath, so that, as detailed in [ML01], every XPath query can be translated into LDAP queries of the form:

Definition 10 (LDAPQL Query).
An LDAPQL Query Q_{HL} is a 4-tuple

$$Q_{HL} = (b_{Q_{HL}}, s_{Q_{HL}}, f_{Q_{HL}}, p_{Q_{HL}})$$

such that:

- *$b_{Q_{HL}}$ is the distinguished name of the base entry in the directory instance where the search starts from.*

- $s_{Q_{HL}}$ *is the scope of the search, which can be:*
  `base` *if the search is to be restricted to just the first node,*
  `onelevel` *if only the first level of nodes is to be searched,*
  `subtree` *if all nodes under the base should be considered by the filter expression,*
  `ancestors` *if all the ancestors of the node up to the root are to be searched.*
- $f_{Q_{HL}}$ *is the filter expression defined as the boolean combination of atomic filters of the form* $(a\ op\ t)$*, where:*
  - a *is an attribute name;*
  - op *is a comparison operator from the set* $\{=, \neq, <, \leq, >, \geq\}$*;*
  - t *is an attribute value.*
- $p_{Q_{HL}}$ *is an (optional) projection of LDAP attributes that define the set of attributes to be returned by the query. If* $p_{Q_{HL}}$ *is empty, all attributes are returned.*

□

Example 4 (LDAPQL Query). The LDAPQL query

$$Q_L = (\text{“cn=Queries,cn=Cache,dc=top”}, subtree, (oc = XMLQuery), \{hash\})$$

retrieves the `hash` attribute from all `XMLQuery` nodes stored under the node with distinguished name “cn=Queries,cn=Cache, dc=top”. □

For the purposes of this paper, it suffices to know that the *eval* predicate explained in the previous sections is implemented by means of generic algorithms that translate an arbitrary XPath expression into an LDAPQL construct and evaluates it, following the serial approach depicted in figure 1.

More specifically, given the nature and structure of the XPath model and of our evaluation algorithm, we can summarize the process involved in the translation and evaluation of XPath queries with the picture represented in figure 3. In it, we can see that each subquery evaluation involves the generation of two types of queries:

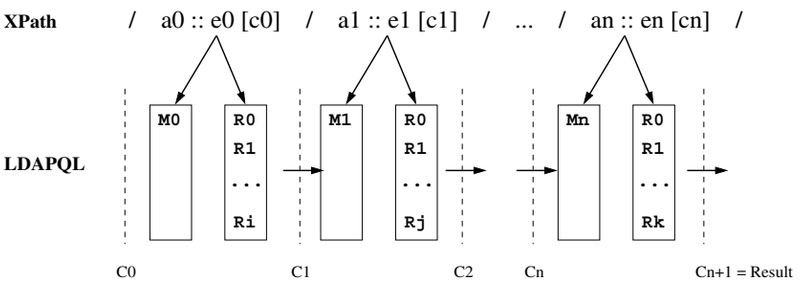


Fig. 3. XPath Evaluation

1. *Main queries*, depicted in the figure by queries $M_0 \ldots M_n$ at each step; and
2. *refinement queries*, represented, for example, by the $R_0 \ldots R_i$ set in the first subquery.

The output context of a specific step is uniquely determined by evaluating the set of main and refinement queries on the input context at each step of the computation. For example, C_2 is computed by evaluating M_1 and $R_0 \ldots R_j$ on C_1.

At any given path step w_i, there is one unique main query M that captures the semantics of the axis a_i and node expression e_i, and a set of refinement queries $\{R_i\}$ that correspond to each boolean predicate in c_i.

Example 5 (LDAPQL Evaluation Example). Given the following query extracted from the Mondial database [May]: $Q_X = /child :: mondial/child :: country[attribute :: car\_code = \text{“}D\text{”}]$, the application of the evaluation algorithm, produces the following results:

- $q_0 = /child :: mondial/$
  - $C_0 = \{dn(root)\}$, since we start at the beginning of the document.
  - $w_0 = child :: mondial[\,]$
  - $C_1 = \{dn(mondial)\}$
- $q_1 = child :: country[attribute :: car\_code = \text{“}D\text{”}]$
  - $C_1 = \{dn(mondial)\}$
  - $w_1 = child :: country[attribute :: car\_code = \text{“}D\text{”}]$
  - $C_2 = \{dn(Germany)\}$, since Germany is the only country in Mondial whose `car_code` attribute has the value “`D`”.

The evaluation of the first subquery w_0, produces the following LDAP queries:

- $w_0 = child :: mondial[\,]$
  - $M_0 = (dn(root), \texttt{onelevel}, (\&(oc = \texttt{XMLElement})(name = \text{“}mondial\text{”})), \{\})$
  - $R_0 = \{\}$

Since $w_0 = child :: mondial[\,]$ has no conditional predicates, the set of refinement queries R_0 is empty.

The evaluation of the second subquery w_1, on the other hand, produces the following queries:

- $w_1 = child :: country[attribute :: car\_code = \text{“}D\text{”}]$
  - $M_1 = (dn(mondial), \texttt{onelevel}, (\&(oc = \texttt{XMLElement})(name = \text{“}country\text{”})), \{\})$
  - $R_1 = \{(dn(country), \texttt{onelevel}, (\&(oc = \texttt{XMLAttribute})(\&(name = \text{“}car\_code\text{”})(value = \text{“}D\text{”}))), \{\})\}$

In this case, the set of refinement queries is not empty because w_1 contains the predicate $attribute :: car\_code =$ "D".

The only query in R_1 is generated by our algorithms because the first term of the equality testing predicate is a path expression, and the second term a simple value (the constant "D"). □

Therefore, the evaluation of XPath queries, independently of the underlying model used for their computation (LDAPQL, DOM [ea00, HHW+00], etc.) consists of two phases that occur at every step:

1. A (possibly) *expansive* phase, represented by the set of *main queries*.
2. A (definitely) *implosive* phase, represented by the set of *refinement queries*.

4.3 Cache Data Model

The core of the cache data representation lies in the specification of the custom-defined `XMLQuery` class in the LDAP directory schema to allow for the storage of the *cache* relation.

Figure 4 contains the complete representation of the `XMLQuery` node, where the `oc`, `hash`, `context`, `scope`, `xpathquery` and `result` attributes are stored.

The meaning of the `oc` attribute has already been defined in the previous section. It simply contains the name of the LDAP class a particular node belongs to. In our case, all nodes used to represent either a query or part of it, have a value of `XMLQuery` in their `oc` attribute.

The `hash` attribute contains the MD5 encoded string [MvOV97] that uniquely identifies a query, and is used to very efficiently determine whether or not there are any `XMLQuery` nodes in the system that contain the previously cached result for a particular set of nodes.

The next four attributes, `context`, `scope`, `xpathquery` and `result` define a query or subquery in terms of the characteristics described in the XPath specification [CD99].

The `context` attribute stores the set of nodes in the input set of the *cache* predicate, and the `result` attribute, a tuple that stores each input node with its

```
XMLQuery OBJECT-CLASS ::= {
  SUBCLASS OF {top}
  MUST CONTAIN {oc,hash,context,scope,xpathquery,result}
  TYPE oc OBJECT-CLASS
  TYPE hash STRING
  TYPE context DN
  TYPE scope STRING
  TYPE xpathquery STRING
  TYPE result (DN, DN)
}
```

Fig. 4. LDAP Class for Query Representation

corresponding output node. In other words, the contents of the `result` attribute is a set of distinguished name tuples (dn_i, dn_j), such that dn_j is the result of applying the query stored in the `xpathquery` attribute under the scope defined in the `scope` attribute on the node dn_i from the `context` of the query.

Finally, the `scope` and `xpathquery` attributes simply contain the human-readable form of the stored query to be used for control and debugging purposes.

4.4 Cache Evaluation Model

Combining the models we have seen so far and the fact that, in our particular implementation of the HLCaches system, we are dealing with an LDAP system, we can redefine what it means for a rewriting to be equivalent, weak equivalent or partial equivalent as follows:

Definition 11 (HLCaches Equivalent Rewriting).
Given an XPath subquery $q_i = (C_i, w_i, C_{i+1})$, an equivalent rewriting is an LDAP node n in an directory instance, such that all of the following properties hold:

1. $oc(n) =$ *XMLQuery*,
2. $C_i = context(n)$, *and*
3. $hash(w_i) = hash(n)$.

The functions $oc(n)$, $context(n)$ and $hash(n)$ simply return the value or values the corresponding attribute has stored at node n.

We further assume that the hash function used to create the entry in the LDAP node and the hash of the path expression w_i normalize the expression before applying the encoding so that equivalent expressions like $a \wedge b$ and $b \wedge a$ are encoded to the same string. □

The evaluation of an equivalent rewriting in HLCaches is thus performed in a very efficient way, since the result is already stored in the `XMLQuery` node.

Definition 12 (HLCaches Equivalent Rewriting Evaluation).
Let q_i, defined as $q_i = (C_i, w_i, C_{i+1})$, be an XPath subquery, and n its equivalent rewriting for a specific directory instance. Then, the evaluation of the subquery, and therefore, the contents of C_{i+1} are $C_{i+1} = result(n)$. □

Following the same approach, we have the following definition for weak equivalent rewritings:

Definition 13 (HLCaches Weak Equivalent Rewriting).
Given an XPath subquery q_i, defined as usual $q_i = (C_i, w_i, C_{i+1})$, a weak equivalent rewriting is an LDAP node n in a directory instance, such that all of the following properties hold:

1. $oc(n) =$ *XMLQuery*,
2. $C_i \subseteq context(n)$, *and*
3. $hash(w_i) = hash(n)$.

The same restrictions and remarks given for the definition of equivalent rewritings hold. □

Given the new definition of weak equivalency, we need to redefine what it means to evaluate and compute the result set of a weak equivalent rewriting, since the contents of the `result` attribute in the rewriting are (possibly) a superset of the required solution.

Definition 14 (HLCaches Weak Equivalent Rewriting Evaluation). *Given an XPath subquery $q_i = (C_i, w_i, C_{i+1})$ and a weak equivalent rewriting n for a specific directory instance, the evaluation of the subquery, and therefore, the contents of C_{i+1} are computed by means of an LDAPQL query, as follows:*

$$C_{i+1} = LDAP(n, \texttt{base}, \{(\&(oc = \texttt{XMLQuery})(result = (C_i, *)))\}, \{result\})$$

□

This definition assumes, as we have already mentioned, that each of the `result` entries in the rewriting are stored in the form of a tuple relation where the first tuple is the individual LDAP node evaluated, and the second one of the possibly many result nodes.

Finally, we can redefine what it means for a rewriting in HLCaches to be partial:

Definition 15 (HLCaches Partial Equivalent Rewriting). *Given q_i, an XPath subquery defined as usual $q_i = (C_i, w_i, C_{i+1})$, a partial equivalent rewriting is an LDAP node n in a directory instance, such that all of the following properties hold:*

1. $oc(n) = \texttt{XMLQuery}$,
2. $C_i \supset context(n)$, *and*
3. $hash(w_i) = hash(n)$.

The same restrictions and remarks given for the previous definitions hold. □

Therefore, and following the same mechanism used for the evaluation of weak equivalent rewritings, we have:

Definition 16 (HLCaches Partial Equivalent Rewriting Evaluation). *Given a subquery q_i, defined as usual $q_i = (C_i, w_i, C_{i+1})$, and a partial equivalent rewriting n for a specific directory instance, the partial evaluation of the subquery, and therefore, partial contents of C_{i+1} are computed by means of the following LDAP query:*

$$C_{i+1} = LDAP(n, \texttt{base}, \{(\&(oc = \texttt{XMLQuery})(result = (context(n), *)))\}, \{result\})$$

□

Algorithm FIND_EQUIVALENT (q_i /* XPath subquery */)

Let q_i be an XPath subquery of the form $q_i = (C_i, w_i, C_{i+1})$
Let *Result* be a set of nodes that holds the result
Let t = "cn=Queries, cn=Cache, dc=top" be the cache top node

/* Perform an LDAP query to find all candidate rewritings */
$Result = \text{LDAP}(t, \texttt{subtree}, (\&(oc = \texttt{XMLQuery})(hash = hash(w_i))), \{\})$

/* Test each candidate for equivalency */
for each candidate $c \in Result$
  if (not $C_i \subseteq context(c)$)
    /* Remove non-equivalent rewritings */
    $Result = Result \setminus c$

return *Result*

Fig. 5. FIND_EQUIVALENT algorithm

Given the set of definitions detailed above, it remains to determine how to find equivalent rewritings in an efficient way. Following the view materialization approach, other researchers have developed rather complicated algorithms that try to achieve this goal [LMSS95, LRO96], although the efficiency of their approaches is not that impressive. The bucket algorithm [LRO96], for example, uses what could be considered a purely brute force approach after performing a quite rudimentary pruning of candidate views. Even after this pruning, the complexity of the bucket algorithm is still $O(|V| \cdot |Q|)$, where $|V|$ is the number of views in the system, and $|Q|$ the size of the query in terms of individual predicates. Furthermore, their approach is tailored exclusively for conjunctive queries, whose expressiveness is definitely a subset of that of XPath queries.

HLCaches, on the other hand, is not limited to conjunctive queries, allows the full expressive power of XPath, and is able to find equivalent rewritings very efficiently at the subquery level. Figure 5 contains pseudocode for the `FIND_EQUIVALENT` algorithm whose purpose is to look for equivalent subqueries applying the previous definitions in an efficient way.

The advantages of such an algorithm in comparison to the classical algorithms that try to find equivalent rewritings is twofold:

1. The search can be performed using just one LDAP query, and a simple subset test to remove partial equivalent rewritings. Given the nature of the MD5 encoding [MvOV97] used to implement the hash function, the number of nodes having the same hash is very limited in practice.
2. The search and evaluation of the rewriting are performed in one step, thus eliminating the need for an extra (costly) evaluation procedure.
3. The nature of LDAP instances, where information is stored in hierarchical trees, allows us to define the "cn=Queries, cn=Cache, dc=top" node to be the root of all stored queries, thus speeding up the process of finding equivalent rewritings.

For illustration purposes, the `FIND_EQUIVALENT` algorithm shown here only accepts equivalent rewritings, but it can be very easily modified to also return partial equivalent rewritings, simply by removing the extra subset checking at the end of the procedure. Such an algorithm, besides providing us with partial equivalent rewritings, also allows us to implement partial evaluation of XPath queries.

5 Conclusion

In this paper, we have provided a formal model to efficiently solve the problem of cache answerability for XPath queries when performed over XML data. The generality of our model is backed-up by the fact that it can be represented and studied independently of the storage model used for XML, but we have also provided examples and details about the implementation of such a model in the context of HLCaches, an LDAP-based distributed caching system developed by the authors for the efficient processing of XPath queries.

The efficiency of our implementation lies on the fact that the underlying representation model (LDAP) is very similar to the storage model defined by XML. Furthermore, the results and implementation details given on our LDAP-based implementation show that the checking, and evaluation of rewritings at the subquery level for XPath expressions can be performed in a very efficient way, as opposed to more classic approaches, where the efficiency of their query containment and query rewriting algorithms depend exponentially on the number of views stored in the system.

References

[BPSMM00] Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, and Eve Maler. Extensible markup language (XML) 1.0 (second edition). http://www.w3.org/TR/2000/REC-xml-20001006, October 2000. 183, 190

[CD99] James Clark and Steve DeRose. XML path language (XPath) version 1.0. http://www.w3c.org/TR/xpath, November 1999. 183, 184, 186, 189, 194

[CGLV00] Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Moshe Y. Vardi. View-based query processing for regular path queries with inverse. In *Proceedings of the Nineteenth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, pages 58–66, Dallas, Texas, USA, May 2000. ACM Press. 183

[DFJ+96] Shaul Dar, Michael J. Franklin, Björn Jónsson, Divesh Srivastava, and Michael Tan. Semantic data caching and replacement. In T. M. Vijayaraman, Alejandro P. Buchmann, C. Mohan, and Nandlal L. Sarda, editors, *Proceedings of 22th International Conference on Very Large Data Bases (VLDB) 1996*, pages 330–341, Mumbai, Bombai, India, Septemter 1996. Morgan Kaufmann. 183

[ea00] L. Wood et al. Document object model (DOM) level 1 specification (2nd ed.). http://www.w3.org/TR/2000/WD-DOM-Level-1-20000929/, September 2000. 194

[Fal01] David C. Fallside. XML Schema part 0: Primer. http://www.w3.org/TR/xmlschema-0/, May 2001. 190

[HHW+00] Arnaud Le Hors, Philippe Le Hégaret, Lauren Wood, Gavin Nicol, Jonathan Robie, Mike Champion, and Steve Byrne. Document object model (DOM) level 2 core specification. http://www.w3.org/TR/2000/REC-DOM-Level-2-Core-20001113/, November 2000. 194

[HSG99] T. A. Howes, M. C. Smith, and G. S. Good. *Understanding and Deploying LDAP Directory Services*. Macmillan Network Architecture and Development. Macmillan Technical Publishing U. S. A., 1999. 189

[KNS99] Yaron Kanza, Werner Nutt, and Yehoshua Sagiv. Queries with incomplete answers over semistructured data. In *Proceedings of the Eighteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 227–236, Philadelphia, Pennsylvania, USA, May 1999. ACM Press. 183

[Lev00] A. Levy. Logic-based techniques in data integration. In J. Minker, editor, *Logic-Based Artificial Intelligence*. Kluwer Publishers, 2000. 183

[LMSS95] Alon Y. Levy, Alberto O. Mendelzon, Yehoshua Sagiv, and Divesh Srivastava. Answering queries using views. In *Proceedings of the Fourteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 95–104, San Jose, California, USA, May 1995. ACM Press. 197

[LRO96] Alon Y. Levy, Anand Rajaraman, and Joann J. Ordille. Querying heterogeneous information sources using source descriptions. In T. M. Vijayaraman, Alejandro P. Buchmann, C. Mohan, and Nandlal L. Sarda, editors, *Proceedings of 22th International Conference on Very Large Data Bases (VLDB) 1996*, pages 251–262, Mumbai, Bombai, India, Septemter 1996. Morgan Kaufmann. 183, 197

[Mar01] Pedro José Marrón. *Processing XML in LDAP and its Application to Caching*. PhD thesis, Universität Freiburg, October 2001. 184

[May] Wolfgang May. Mondial database. http://www.informatik.uni-freiburg.de/~may/Mondial. 188, 193

[ML01] Pedro José Marrón and Georg Lausen. On processing XML in LDAP. In *Proceedings of the 27th International Conference on Very Large Data Bases (VLDB)*, pages 601–610, Rome, Italy, September 2001. Morgan Kaufmann. 184, 189, 191

[MvOV97] Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1997. 194, 197

[PV99] Yannis Papakonstantinou and Vasilis Vassalos. Query rewriting for semistructured data. In Alex Delis, Christos Faloutsos, and Shahram Ghandeharizadeh, editors, *Proceedings of the ACM SIGMOD International Conference 1999*, pages 455–466, Philadephia, Pennsylvania, USA, June 1999. ACM Press. 183

[QCR00] Luping Quan, Li Chen, and Elke A. Rundensteiner. Argos: Efficient refresh in an XQL-based web caching system. In *Proceedings of the Third International Workshop on the Web and Databases*, pages 23–28, Dallas, Texas, May 2000. 183

[TBMM01] Henry S. Thompson, David Beech, Murray Maloney, and Noah Mendelsohn. Xml schema part 1: Structures. http://www.w3.org/TR/xmlschema-1/, May 2001. 190

[WHK97] M. Wahl, T. Howes, and S. Kille. Lightweight directory access protocol (v3). RFC 2251, December 1997. 189

Web-Based Integration of Printed and Digital Information

Moira C. Norrie and Beat Signer

Dept. of Computer Science, ETH Zurich
CH-8092 Zurich, Switzerland
{norrie,signer}@inf.ethz.ch

Abstract. The affordances of paper have ensured its retention as a key information medium, in spite of dramatic increases in the use of digital technologies for information storage, processing and delivery. Recent developments in paper, printing and wand technologies may lead to the widespread use of digitally augmented paper in the near future, thereby enabling the paper and digital worlds to be linked together. We are interested in using these technologies to achieve a true integration of printed and digital information sources such that users may browse freely back and forth between paper and digital resources. We present web-based server technologies that support this integration by allowing users to dynamically link areas of printed documents to objects of an application database. The server component is implemented using the eXtensible Information Management Architecture (XIMA) and is independent of the particular paper, printing and reader technologies used to realise the digitally augmented paper. The framework presented manages semantic information about application objects, documents, users, links and client devices and it supports universal client access.

1 Introduction

The issue of architectures and technologies for the integration of existing data sources has been a topic of interest for many years within the database community. A number of solutions have been proposed with the approach taken dependent on the degree of heterogeneity, the level of local autonomy and, of course, the required functionality of the target system. In terms of heterogeneity, many forms of data sources have been covered including different categories of database management systems, knowledge base systems, file systems and web documents. But one major form of data source has so far been neglected, namely paper documents.

Predictions of the paperless office no longer seem realistic as indicated by the continued increase in office paper consumption over the last two decades [29]. Digital technologies have advantages in terms of storing and accessing large amounts of information, displaying multimedia and fast full-text searching. Further, it is a dynamic medium that enables both contents and links to be updated with ease. But paper technology has its own advantages over digital media in

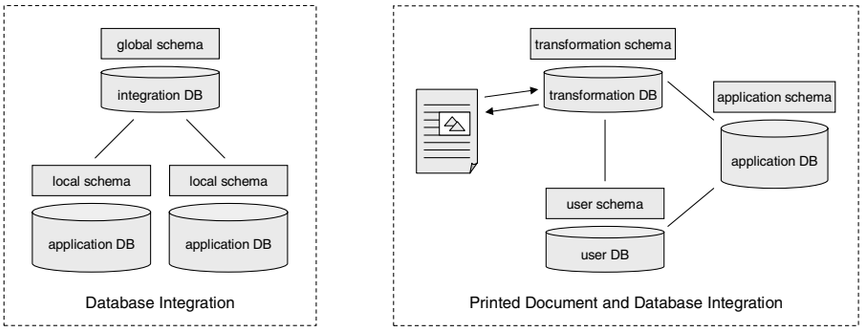
S. Bressan et al. (Eds.): EEXTT and DIWeb 2002, LNCS 2590, pp. 200–219, 2003.

terms of navigation within and across documents. It is much simpler to scan a book by rapidly flicking through pages than to browse through a digital document [24]. Paper also supports forms of collaboration and interaction that are difficult to mimic in current digital environments [19]. Ideally, we would like to achieve an integration of paper and digital technologies, thereby gaining the best of both worlds.

While a number of projects have proposed different variants of digitally augmented paper as a means of linking printed and digital media, the focus has been on the paper, printing and wand technologies rather than on the data integration concepts. As a result, the linking mechanisms tend to be based on physical rather than logical resources. For example, a physical position within a document may link to a particular multimedia file. The result is that links are uni-directional and static. We however link at the level of logical resources, enabling links to be dynamic, context-dependent and multiple. It should not only be possible to link from paper to digital resources, but also to link from digital resources to paper and even from paper to paper. In this way, we are able to achieve an integration of printed and digital information rather than simply of the underlying media. Further, users are able to freely author and share links, thereby enhancing the traditional qualities of paper as a static medium with dynamic linking features usually associated with digital media.

The key to semantic integration of printed and digital information lies primarily with the server rather than the client technologies. A suitable framework must be capable of dynamically mapping document positions to information objects and vice versa. It must manage not only the links, but also their semantics and have flexible means of classifying and associating both logical and physical resources. In addition, it must allow for managing information about both users and client devices. Finally, the framework must have a clear separation of content and presentation, enabling documents to be generated dynamically to suit client device, user and context. We have developed such an integration server in the context of a European project called Paper^{++} [26] which is developing and assessing innovative concepts and technologies aimed at enriching the use of paper in everyday settings. In line with many other current projects on data integration, we use the web as our basic integration platform.

In Sect. 2, we introduce our system and discuss how it relates to traditional data integration architectures. Section 3 then describes the general technologies for digitally augmented paper that underly our approach. In Sect. 4, we explain the overall architecture in terms of the relevant client and server components. Details of the XIMA and OMS data management components used to implement the integration framework are presented in Sect. 5 and 6, respectively. A discussion of related work is given in Sect. 7. Finally, concluding remarks are presented in Sect. 8.



(a) (b)

Fig. 1. Data integration architectures

2 Integrating Printed and Digital Data

As stated in the introduction, the details of particular data integration architectures depend on many factors such as the requirements of the target system and the characteristics of the data sources in terms of heterogeneity and autonomy of operation. Earlier projects focussed on the specific problem of database integration e.g. [10, 30]. More recent research has tended to address the more general problem of integration across many forms of information sources inclusive of databases, knowledge bases, digital libraries and web documents as described in [5, 35]. While these architectures may vary greatly in terms of complexity, scope and approach, what they have in common is the use of some form of semantic integration component, be it a global schema, mediator or metadatabase.

For example, in Fig. 1.a, we show a simplified version of a typical database integration architecture where the local schemas are integrated into a global schema. Associated integration data may be stored at the global level to perform the necessary mappings between global semantic concepts and those of local application databases.

When we want to integrate paper information sources with digital information sources, the situation is different in that there is no schema for a paper information source, i.e. there is no available metadata. The integration of information printed on paper and that of a database must be performed by linking documents, pages and parts of pages to information objects of the application database. As shown in Fig. 1.b, this is done through a transformation database which can map physical document positions to database objects and vice versa.

To obtain the required flexibility of being able to map physical document positions to multiple database objects and for that mapping to be dependent on both user and context, we have introduced the concepts of *shapes*, *layers* and *users* into our model. A shape describes a selectable area of a page. Generally, a digital wand (barcode reader) will return a coordinate position and the trans-

formation database maps this to the corresponding shape, which is, in turn, associated with one or more objects of the application database.

Every shape is positioned on a specific layer. In the case that a point lies within the intersection of two or more shapes, the information object which is bound to the shape at the topmost layer of the set of active layers is returned. Furthermore, it is possible to selectively activate and deactivate specific layers which enables us to generate context dependent results by binding a particular position on a page to different resources according to the current active layer set.

More formally, assume that there are k layers numbered $1, 2, \ldots, k$, and that each shape S_i is associated with exactly one layer i.e. $\text{layer}(S_i) = j$ where $1 \leq j \leq k$.

For a given point (x, y), we write $(x, y) \in S_i$ to denote that the point (x, y) is contained within the shape S_i. If two shapes S_i and S_j are overlapping, i.e. there exists some point (x, y) such that $(x, y) \in S_i$ and $(x, y) \in S_j$, then the shapes must be on different layers i.e. $\text{layer}(S_i) \neq \text{layer}(S_j)$.

At any point in time, the set of active layers is a subset of all layers. Let A denote the set of all active layers. Then $A \subseteq \{1, 2, \ldots, k\}$.

For a set of shapes $S = \{S_1, S_2, \ldots, S_n\}$, the selected shape associated with the given point (x, y) is given by the function s

$$s(x, y) = S_m$$

where

$$\text{layer}(S_m) = max\{j | S_i \in S, (x, y) \in S_i, \text{layer}(S_i) = j \text{ and } j \in A \}$$

Figure 2 illustrates how the layering concept is used to annotate a page with links to digital information. On the left we see the physical page whereas the right side shows the page's representation within the interaction database. The current position P lies within the rectangular shape on layer 5 (defining Anchor 1) as well as in the shape on layer 10 which is defined by the reptile (Anchor 2). As explained above, in the case of any ambiguity the shape on the topmost layer always has the highest priority. Therefore the link bound to Anchor 2 which is defined by the reptile's shape will be chosen.

Note that the layers should not be "misused" to deploy personalised content to different users (e.g. by assigning a specific range of layers to every user and only activating the user specific layers). Rather, the system built-in user concept should be used to deploy user dependent content.

The user data component shown in Fig. 1.b manages information about the different users of the system. A user can define his preferences and can also be classified into different types of user groups. This enables an author of a digitally augmented document to, not only define different information anchors for different users, but also to bind different content to an information anchor based on the same user information. Therefore, we can use the same augmented paper to deploy different information based on a user's profile. A typical application of this

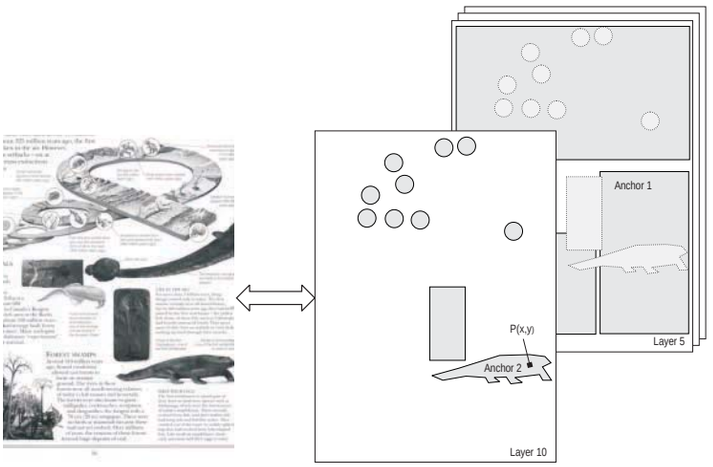


Fig. 2. Multi layer authoring

user context-dependent information delivery is in an educational environment, where a teacher gets additional information for the same resource anchors. For example, a teacher may receive the solutions for exercises included in the course materials, whereas a student only obtains hints or links to other course material. The same user preferences can be used for different applications, i.e. a user does not have to specify new preferences for each application. For this reason, we show the user database in Fig. 1.b as a separate component.

Having introduced the basic idea of integrating paper and digital information sources, we now go on to discuss the details of technologies and architectures required to achieve this integration.

3 Digitally Augmented Paper

As described in the previous section, the integration of printed and digital information sources is based on the ability to detect page positions within a document. While the paper, printing and reader technologies required to fully support digitally augmented paper are still in their infancy, various forms of wand devices (e.g. barcode readers) are now emerging in the marketplace. Expected developments in the near future in terms of cheaper reading devices and means of invisibly encoding document positions on paper will surely have a dramatic effect on the spread and use of digitally augmented paper.

To annotate physical paper, we first need a technique to identify active areas of a sheet of paper. An active area is any shape defined by the author and might correspond to a paragraph, a text box, a single word, a figure or even part of a figure, etc. Further, active areas may be overlapping, for example, a specific word within a paragraph, text overlapping an image, or part of an image and

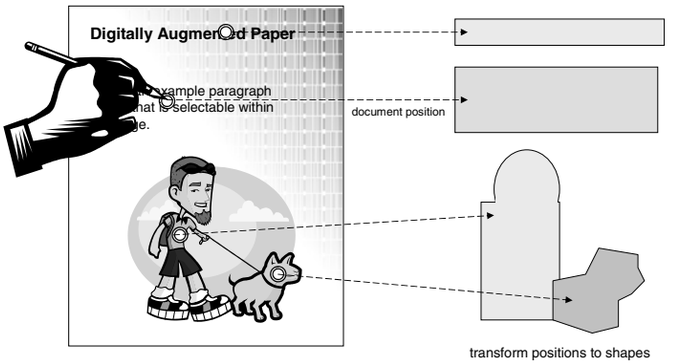


Fig. 3. Mapping of page positions to defined shapes

the whole image. Ideally, a user should be able to point to an active area with a wand and trigger an associated action such as the display of a digital media file or a link menu, or the execution of some application system operation.

In the Paper^{++} prototype, detection of positions within a page is realised by printing on standard paper a grid of barcodes containing information about the page number and the corresponding position in terms of x and y coordinates using an inductive invisible ink. By pointing at the paper with a wand device which is capable of measuring differences of inductivity, we are able to read the barcodes and finally to decode the corresponding page number and the actual position of the reader.

We illustrate the basic idea of our system in Fig. 3. The shaded pattern across the page indicates the invisible barcodes that encode document positions in terms of coordinates. Logical elements within the page can be made selectable by defining corresponding logical shapes. The digital wand will return document positions and the server can then map these positions into corresponding shapes, which in turn are linked to digital information objects.

For example, Fig. 3 shows a page containing a heading, a text paragraph and an image. The author has specified four selectable elements which are the heading, the text, the man within the image and the dog within the image. Each selectable element is specified by a shape as indicated on the right-hand side of the figure. Shapes may be of various forms such as rectangles, polygons or complex shapes such as the one shown for the man, which is composed from an ellipse and a rectangle. Each of these shapes is then associated to one or more objects of the application database. For example, the man's shape may be linked to a person object giving details of that person or possibly further digital multimedia resources such as images, videos or sound.

Note that, in our approach, we are not only capable of linking printed information to digital information, but also to dynamically link printed documents together. This is illustrated in Fig. 4. The left-hand side shows the page of the previous figure, which we can assume has an indirect link to an object for a par-

ticular breed of dog in a nature database. In the nature database, we assume that information is represented about various breeds of animals with associations to the countries in which these breeds can be found. On the right-hand side of the figure, we show an atlas and we further assume that someone has previously authored links from this atlas to the various countries represented in the nature database. In this way, links can be derived from one printed information source to another using an inverse mapping of digital content to pages. For example, a user pointing to the dog in the paper document on the left, could be presented with various links to information, including a link to the atlas document and the position within the document where information about the country from which that breed of dog originates can be found. It is important to realise that here the link from the image of the dog to the map within the atlas has not been explicitly specified by an author. Rather it is dynamically derived based on authored links to concepts of the nature database and the associations within this database.

With the notion of digitally augmented paper that we have presented, it is possible for users to browse back and forth between printed and digital information and also between different sources of printed information. This may contribute to avoiding problems of mode-locking, where a user switching from paper to digital information locks into the digital world and is distracted from the original document. It could be said that our approach not only allows the digital annotation of physical objects, but also the "physical annotation" of digital objects.

Even without any additional multimedia content, our system will provide a powerful tool allowing users to classify and build links between physical documents. Again, we stress that back references to printed documents are not just another piece of text within the database giving for example a book title and a page number. All links back to physical paper are built dynamically based on meta information about other documents bound to the same information object.

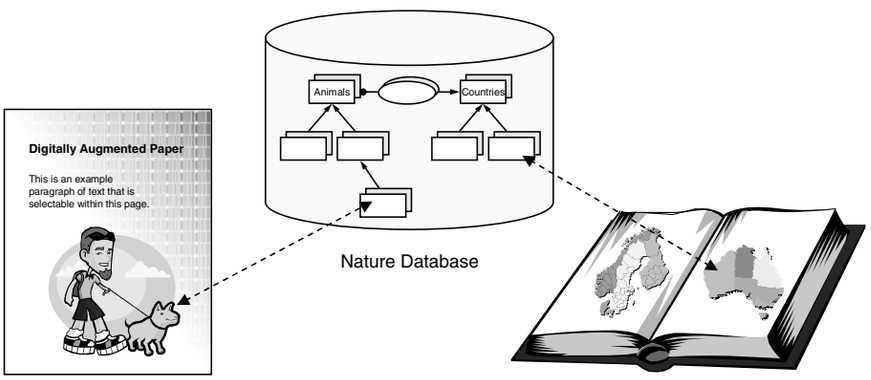
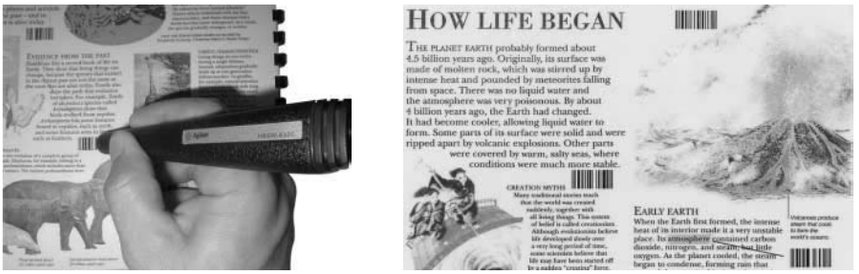


Fig. 4. Linking paper documents

This meta information is already present in the information model (bidirectional associations) and therefore no additional authoring is required.

An important issue is how best to provide users with feedback concerning the location of active areas within a document and also the, possibly context-dependent, associated actions. Whereas the location of the information anchors is obvious in the case of visible barcodes (see Fig. 5), a mechanism will be required to inform users of the presence of active areas when using invisible barcodes. In an authored document, this can be done by highlighting those parts of a page containing links to digital information (as indicated in Fig. 5.b with the highlighting of the word that is the active area corresponding to the barcode in the lower right corner). For specific applications, it may even make sense to omit any visible hints about additional information and leave the task of finding the links completely to the user (e.g. in a children's book where it may be part of a learning task to find additional information such as realised in the LeapPad suite of books [18]). In the case of user generated links, the simplest way to visualise the information anchors is to combine the wand device with a marker pen allowing the user to highlight areas on paper while adding a new link. Further, we have experimented with the use of sound as a possible feedback mechanism.

The development of printing technologies, barcode schemes and wand devices on which such systems could be based is very much an active area of current research. Solutions do exist, but at present they tend to be expensive and based on visible encoding patterns. However, it is expected that cost-effective solutions which are almost invisible and with barcode schemes that provide a good level of position granularity will be available in the near future. Therefore an important factor in the development of our server was to ensure that it is independent of any particular technologies and encoding schemes. What we assume is simply the delivery of a document position. In our discussion of related work in Sect. 7, we describe some alternative technologies to those presented above.



(a) (b)

Fig. 5. Annotated nature encyclopedia page

4 Paper^{++} Server

Our framework is based on a classic client server architecture. On the client side, we have the wand device enabling us to read the document positions using inductive barcodes printed on digitally augmented documents. The client device transmits all of its data to a server which then does further processing of the information. The Paper^{++} server consists of three main components which are shown in Fig. 6. The barcode decoder and visualisation component is responsible for both decoding the barcodes received from the input device and then later presenting the results returned by the eXtensible Information Management Architecture (XIMA) component after processing the request. All data is managed by the interaction and application component which is based on the OMS Java data management system [15, 16].

In a typical Paper^{++} user session, the pointing device transmits recognised barcode information in a preprocessed ASCII format to the barcode transformer, which then decodes relevant information such as the page number and the x and y positions from the barcode. In the next step, an HTTP request including this information about the wand device's current position on the paper document is sent to the XIMA server component. Based on the interaction model and its transformation data, the server fetches the corresponding information from the application database and returns the result in the preferred format of the digital presentation device. For example, a regular web browser would receive an HTML formatted result page, while a WAP-enabled phone would receive a WML result page for the same request.

In Fig. 6, the digital presentation device is a laptop computer which also serves as the barcode transformation component. However note that, in theory, it is not necessary that the barcode transformation and result visualisation take place on the same device. This means that in a public space, for example, the result could be displayed on a large global display screen, while the transformations take place on users' individual portable devices.

It is important to note that all operations on the server side are independent of a particular paper, printing or digital wand technology. Our initial prototype application has been developed using existing barcode reader technologies,

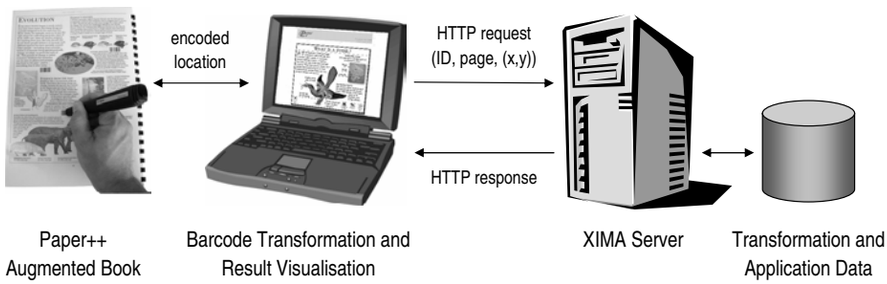


Fig. 6. Paper^{++} architecture overview

(a) (b)

Fig. 7. Nature encyclopedia application running on a laptop and on a Pocket PC

but the barcode transformation process can easily be adapted to any form of device capable of detecting the position on a sheet of paper. All other components are completely independent of the form of input device, although certain technologies may be able to support extended functionalities in terms of user interaction. In Fig. 7.a, we present the configuration that has been used for our initial user studies [8] — the nature encyclopedia application running on a laptop computer (Windows 2000 operating system) and an off-the-shelf barcode reader (Agilent HBSW-8300) to handle input processing. The same application running on an iPAQ H3660 Pocket PC (Windows CE 3.0 operating system) equipped with an 802.11b WLAN card is shown in Fig. 7.b.

To remain flexible in handling user input, we defined the entire logic of the input device in separate interfaces. Through a simple replacement of the concrete Java class implementing the wand device interface, we can therefore easily switch to different input devices.

The XIMA server and the OMS Java system are general components for access to and the management of data. In the case of the Paper^{++} server, they not only manage all the transformation, user and application data, but also perform the necessary request processing inclusive of mapping document positions to application database objects and generating client-specific presentations of data. In the following sections, we describe first the XIMA framework, which can be considered as a universal web-based access layer for OMS Java, and then follow this with some comments about the OMS Java data management system in general and the Paper^{++} server databases in particular.

In Tab. 1 we provide a brief summary of the major requirements to achieve a true integration of printed and digital information sources and our architectural choices to fulfil those requirements. Some of these solutions and technologies (e.g. the eXtensible Information Management Architecture or the OM object model) will be described in detail in the remaining sections.

Table 1. Architectural Choices

Requirement	Architectural Choice
Integration of different heterogeneous information sources.	Web-based client server integration platform.
Context dependent content delivery based on user profile, output format preferred by client device, etc.	eXtensible Information Management Architecture (XIMA) for universal client access to information.
Support for four types of links (paper-to-digital, digital-to-digital, digital-to-paper and paper-to-paper).	Bidirectional association from resource anchors to information resources enabling inverse mapping of resources to pages.
Active page areas (instead of simple barcode mapping) and flexibility in defining the granularity of those active areas.	Transformation database introducing concepts of shapes to define active areas and layers to improve flexibility.
Support for links to web resources as well as to semantically rich database application objects.	OM object model with its rich object classification and high level association construct.
Flexibility in supporting different kinds of input devices and various kinds of position encodings.	Extensible input processing and transformation components based on a small set of interfaces.

5 XIMA Framework

The eXtensible Information Management Architecture (XIMA) was developed to support universal client access to information systems based on web technologies [31]. Many existing tools for web site engineering either rely on a specific form of client such as an HTML browser, or they require that different forms of clients be handled separately meaning that it requires significant development time to port an application to another form of client device. We wanted a universal client framework that required minimal development effort to access web documents via another form of client device. Any additional effort would rather go into optionally customising the presentation to suit the device in question. Such general frameworks are particularly important when one considers how dynamic the world of mobile devices currently is.

An overview of the XIMA server component is given in Fig. 8. For a specific application, all client access is via a single Java servlet — the Entry Servlet. This means that only a single URL needs to be memorised for a specific web site, rather than having different URLs for different types of clients. The Entry Servlet detects the user agent type from the HTTP request header and delegates the handling of the request to the appropriate servlet. For example, we show in Fig. 8 servlets to handle requests from HTML browsers, XML browsers and also WAP phones in terms of WML browsers. In addition, we support access via i-mode mobile phones and regular speech telephones based on the speech recognition and synthesis support provided by VoiceXML servers.

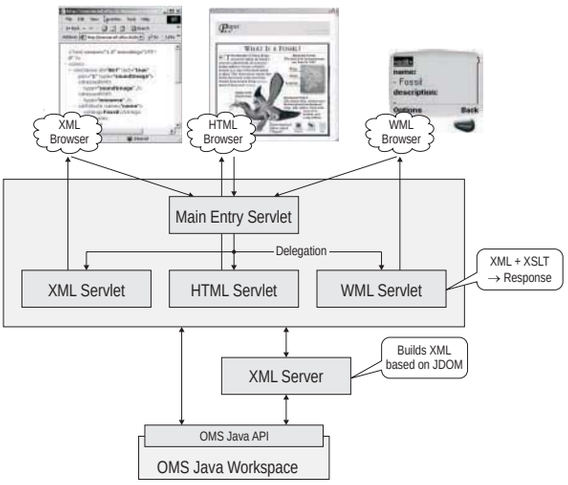


Fig. 8. XIMA server component

The request handling servlets then access the database by connecting to an OMS Java workspace via the OMS Java API. The connection may either be direct or via the XML server component. Direct connections deal with requests that do not involve data retrieval such as checking the membership of an object in a collection or performing updates. Any requests that involve the retrieval of data go through the XML server which forwards them to the OMS Java data management system and generates XML representations for any data objects to be returned to the servlets. The requesting servlets then use the appropriate XSLT templates to transform the XML results to the corresponding responses to be returned to the client.

A point to note in this general architecture is that we are not storing any XML documents, but rather generating them dynamically from the application data which is stored according to the information model. Since the information model is a loosely-coupled graph model based on object collections and associations, this gives much more flexibility than the rather restrictive hierarchical-based models imposed by XML structures. At access time, a particular hierarchical view on data is derived and the appropriate XML structure generated.

Using generic XSLT templates for the various client devices, we are able to provide generic browsers and editors for the current set of client device types. Adding a new type of client device involves implementing the corresponding servlet and writing the appropriate XSLT templates. Specific application interfaces are supported through the customisation of XSLT templates and the specification of document structures in terms of how document content is built dynamically from one or more individual information objects. It is beyond the scope of this paper to describe all the details of the XIMA framework, but further information can be found in [31].

6 OMS Data Management Component

When discussing the general integration architecture in Sect. 2, we showed three database components — the transformation database, the user database and the application database. In practice, these may all be within a single database or they could be separate databases. For example, the user preferences may be general and apply to more than one application and, hence, we could choose to store them in a separate database and would use this in conjunction with a number of application database components. If the application database exists already, then it makes sense to perform the integration through a separate transformation database — especially if the application actually involves a number of application databases (or even other forms of digital information source). In the actual prototype systems that we have built so far for Paper^{++}, the databases were created from scratch and hence we integrated the components within a single database. However, it would be no problem to distribute the components and we even have a version of OMS that specifically supports peer-to-peer connection among distributed database components [22].

The transformation, user and application data is stored using the OMS Java data management framework which is based on the OM object model [23]. OM is a generic, object-oriented model that supports information modelling through a two-level structure of classification and typing. Entities are represented by objects with attributes, methods and triggers as defined by the associated object types. The classification of entities is done through semantic groupings represented by named collections with a specified member type. The member type of a collection not only constrains its membership, but also specifies a contextual view of objects accessed through that collection. Thus, we can have role-dependent object views and behaviours by associating different types with collections according to the roles that they represent.

OM has a high level association construct which enables relationships to be classified and manipulated directly. The association construct further supports the decoupling of information objects since objects tend to be referenced through the association concept, rather than through object reference attributes. Last but not least, the model includes a powerful set of operations over objects, collections and associations.

The expressive features of the OM model enable us to capture the semantics of application domains in terms of a simple, but powerful set of constructs. Its support for the direct representation and manipulation of associations is particularly useful in supporting link management in systems that support hypermedia browsing such as Paper^{++}.

In Fig. 9, we show part of the application database model for the nature database (already mentioned in Sect. 4) that was developed for a Paper^{++} prototype to integrate printed and digital information associated with a nature encyclopaedia. As can be seen in the model, all objects of the application model are *resources*. This means that they can be associated with anchors defined in printed documents. Using the *relatedTo* association, each resource can further be linked to other resources. The system still allows links to be built to simple

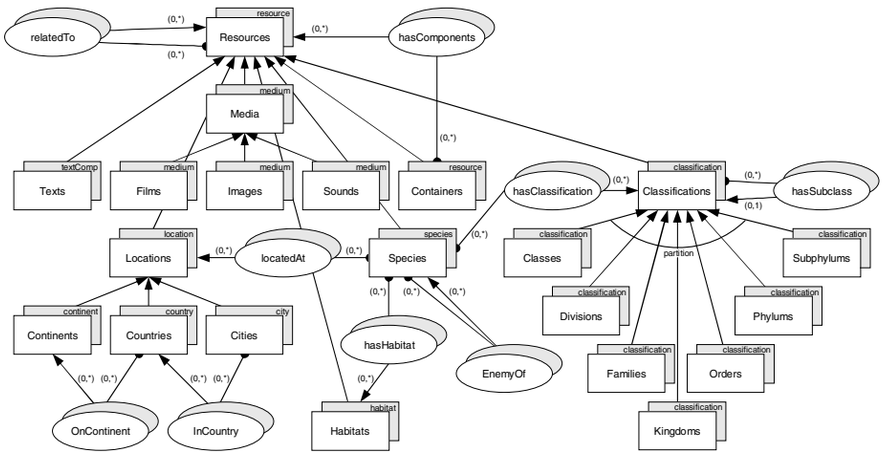


Fig. 9. Nature encyclopaedia application model

multimedia objects such as pieces of text, images or sound clips as shown in the upper part of the figure. However, it becomes more interesting if we start linking parts of a physical document to so-called *application objects* representing concepts of a specific application domain.

In the case of the nature database, we have the concepts of species, habitats, different classifications, locations etc. Further, these concepts are connected by specific associations (e.g. *hasClassification*, *hasHabitat*). In this way, we can express the fact that a species is found at a specific location, and we can give each species a classification and add other domain specific semantics. Let us now assume that a user is reading the printed version of the nature encyclopaedia and he wants to get more information about a particular species. By following the link to the digital media, he gets additional information about the animal and can browse the digital information. However, as described earlier in the paper, he can also follow links back to the paper world such as an atlas giving information about the country in which a species is found.

To bridge physical paper and the digital medium, the transformation model defines how a document's meta information is represented. As shown in Fig. 10, a document can contain multiple pages and each page can contain multiple resource anchors (active areas). The actual information anchors are defined by shapes such as rectangles, circles or polygons. The transformation data is used to check whether, for a specific position on a document's page, there exists one or more information anchors, i.e. it allows the system to test if a point lies within one or more shapes. Finally, every resource anchor is associated with an information resource of the application database. We have to point out that such an association is always bidirectional, which means that later we can, not only fetch the information associated with a specific shape, but also get the locations

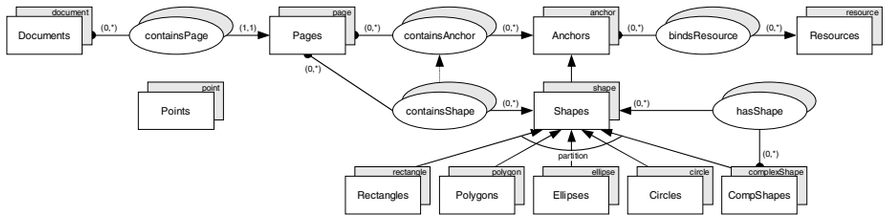


Fig. 10. Transformation database component

of all active areas of printed documents associated with a digital object stored in the application database.

Although not shown in Fig. 10, the model also associates shapes with layers. This enables overlapping shapes and the possibility to link particular parts of a shape to different resources. In the case that a position lies in more than one shape, the one on the topmost, active layer is returned as described in Sect. 2. It would be possible for example to use this to provide a form of "zoom in" by deactivating the topmost layer each time the user points to a document element.

Last but not least, the presentation data model contains information about supported types of client devices and templates for the final rendering of generic XML results. Based on a dynamically generated XML representation of the objects stored in the OMS Java system, the client specific rendering is realised using the Extensible Stylesheet Language Transformation (XSLT). This allows us to not only support a variety of client devices, but also to enable variations in visualisation for the same type of device. A regular user of the application can customise the user interface by defining his own stylesheets, whereas an application can already provide different visualisations for specific groups of users defined by the user model.

7 Related Work

As we will describe in this section, there exist numerous different approaches to encode information on a piece of paper. However, we believe that the power of digitally augmented paper strongly depends on the functionality of the server component handling the requests and not only on the client technologies. Users should be able to freely browse back and forth between the paper and digital worlds, dynamically creating not only annotations, but also links between paper and digital resources, or even between paper resources. In this way, paper would become a *first-class citizen* within information systems. The key to achieving this vision lies in using powerful database technologies on the server side that not only provide flexible and powerful dynamic link management, but also manage application semantics. To explain further what we mean by this, we describe the linking processes supported by existing projects and their limitations.

We start by comparing our system with the *Intelligent Paper* approach described in [6]. They propose a similar solution where standard sheets of paper

are entirely covered with invisible printed marks. In terms of the position detection mechanism, the two approaches are quite similar, although there are some differences in how information is encoded within the barcodes. Instead of using globally unique identifiers on every page, we separate the document identifier from the page identifier and only encode the page identifier within each page of a specific document. However, the major difference between our approach and theirs is in how the mapping from the physical location on paper to the digital content is managed. The *Intelligent Paper* project proposes to use the Adobe Acrobat suite of products to build an electronic counterpart of the paper document with embedded links for every physical page. In contrast, our solution retains the two separate paper and digital worlds and has no need of an equivalent digital version of the physical page. Rather it creates logical links between the two media allowing elements on a physical page to be mapped to the appropriate digital objects.

Using visible barcodes to encode information on a piece of paper is just one possible solution. Another interesting approach from Xerox is based on DataGlyphs [9, 14] to store data on physical paper. They use special cover sheet forms containing DataGlyphs to build paper user interfaces [13]. The cover sheet is placed in front of a document in a scanner or a fax machine connected to a document services system and allows the definition of specific actions to be taken on the document which follows it. However, the augmentation is not on the document itself but rather on an additional piece of paper.

In [28] CyberCode tags (visual 2D-barcodes) are used to identify objects using low-cost CMOS or CCD cameras found increasingly in mobile devices. A drawback of their described potential applications is the limitation to unidirectional links from paper to digital information.

Another technology developed by a Swedish company called Anoto [1] uses a special pattern of tiny dots to print a grid over a page. The corresponding digital pen (e.g. Sony Ericsson's Chatpen) uses a camera to track movement. However, the main focus of the Chatpen and other electronic pens [12, 25] is to digitise information and transmit it to a client device, whereas our goal is to digitally augment the physical medium.

An alternative possibility to create information anchors on physical paper is described in [2] where a highlighter pen is augmented with a small camera. Simple computer vision and pattern recognition techniques allow a user to make marks on paper. However, the stored patterns are not associated with any document meta information which makes it difficult to support bidirectional links enabling a user to browse back and forth between physical paper and digital media.

In [20, 21, 34], a video camera is mounted above an office desk pointing down at the work surface. Additionally a projector is used to display objects on the table. A drawback of these solutions is that the paper always has to be processed on a digital desk and therefore we lose one of the main benefits that paper affords in terms of its mobility. Existing applications which are based on the digital desktop metaphor (e.g. an augmented version of *Alice's Adventures in Wonderland* [4]) do not allow browsing from the digital to the physical medium.

Instead of using a visual annotation to identify a document, Radio Frequency Identification (RFID) tags and the corresponding readers can be used as described in [33]. The RFID tags enable documents or individual pages to be identified: They do not allow specific parts of a page to be addressed. A combination of technologies is presented in [3] where electronic field sensors located in the book binding sense the proximity of the reader's hands and control audio parameters, while RFID tags embedded in each page allow page identification. As described earlier, within the Paper^{++} project, we separate the document identifier from the page identifier. Whereas the identification of pages and elements within a single page is done by using the concept of invisible barcode grids, the idea is to use RFID tags to identify whole documents (e.g. a single book). Alternatively, Infrared Beacons [27] or iButtons [11] can be used to uniquely identify documents as described for example in [7].

Numerous existing projects, e.g. [2, 17, 32], only allow links to single objects such as a piece of electronic text, a sound or a movie clip. They have no support for following links to other related digital resources or back to other paper resources. Thus, the focus of these projects is the ability to annotate a simple sheet of paper with multimedia content. On the server side there is a simple database (in reality often simply a file), that maps positions (more often just single barcodes) to specific digital resources. The processing of a request consists merely of displaying the digital resource.

8 Conclusions

We have described an approach to digitally augmented paper that enables paper to be fully incorporated into digital information spaces as a browsing device. In contrast to existing projects for digitally augmented paper, our approach is based on dynamic links between logical elements of the information sources rather than on hard-coded links between physical media. The key to achieving this was to use a database approach and exploit database concepts in terms of information abstractions, metadata, dynamic links and document generation.

The integration platform was implemented using the existing XIMA object-oriented data management framework, which supports universal client access. In the context of the Paper^{++} project, we have developed two applications — one is the nature encyclopaedia and the other deals with information about the art world. In addition, for use within the project itself, we have developed an application that links information about related technologies, publications and contact persons. The application enables references within printed documents to be linked to other documents (digital or printed), relevant web sites and to information objects within a project database.

These prototype systems are based on conventional barcode technologies, which, although not ideal, have at least been sufficient to test our models and also to carry out initial user studies. Since the server was developed with generality in mind, it will be a simple matter to change to new paper, printing and wand technologies as they emerge in a reliable and cost-effective form.

Links are bidirectional and enable users to move freely back and forth between paper and digital media. This means that we can combine the features of paper (static, excellent readability properties, flexible, inexpensive, ubiquitous) with those of a digital medium (dynamic, can store a lot of information) and, using current skills, have optional access to new functionality.

Acknowledgements

The work reported in this paper is part of the European project Paper^{++}, under the Disappearing Computer Programme (IST-2000-26130). We thank the other members of the project for their contributions and feedback on our work, including Guy Adams, Wolfgang Bock, David Frohlich, Christian Heath, Peter Herdman, Paul Luff, Rachel Murphy, Abigail Sellen and Ella Tallyn.

References

[1] Anoto AB, http://www.anoto.com. 215
[2] T. Arai, D. Aust, and S. Hudson. PaperLink: A Technique for Hyperlinking from Real Paper to Electronic Content. In *Proceedings of ACM CHI'97, Conference on Human Factors in Computing Systems*, March 1997. 215, 216
[3] M. Back, J. Cohen, R. Gold, S. Harrison, and S. Minneman. Listen Reader: An Electronically Augmented Paper-Based Book. In *Proceedings of ACM CHI'2001, Conference on Human Factors in Computing Systems*, Seattle, USA, March 2001. 216
[4] H. Brown, R. Harding, S. Lay, P. Robinson, D. Sheppard, and R. Watts. Active Alice: Using Real Paper to Interact with Electronic Text. In *Proceedings of EP'98, 7th International Conference on Electronic Publishing, Document Manipulation, and Typography*, April 1998. 215
[5] S. Chawathe, H. Garcia-Molina, J. Hammer, K. Ireland, Y. Papakonstantinou, J. Ullman, and J. Widom. The TSIMMIS Project: Integration of Heterogeneous Information Sources. In *16th Meeting of the Information Processing Society of Japan*, 1994. 202
[6] M. Dymetman and M. Copperman. Intelligent Paper. In *Proceedings of EP'98, 7th International Conference on Electronic Publishing, Document Manipulation, and Typography*, April 1998. 214
[7] T. Kindberg et al. People, Places, Things: Web Presence for the Real World. In *Proceedings of WMCSA'2000, 3rd IEEE Workshop on Mobile Computing Systems and Applications*, Monterey, USA, December 2000. 216
[8] D. Frohlich, E. Tallyn, N. Linketscher, B. Signer, and G. Adams. Reading Augmented Paper: Children's Experiences from a Simulation Study. Technical Report HPL-2001-308, HP Labs, 2001. 209
[9] D. L. Hecht. Printed Embedded Data Graphical User Interfaces. *IEEE Computer*, March 2001. 215
[10] D. Heimbigner and D. McLeod. A Federated Architecture for Information Management. *ACM Transactions on Office Information Systems*, pages 253–278, 1985. 202

[11] Dallas Semiconductor, iButton, http://www.ibutton.com. 216
[12] InMotion, e-pen, http://www.e-pen.com. 215
[13] W. Johnson, H. Jellinek, L. Klotz, Jr., R. Rao, and S. Card. Bridging the Paper and Electronic Worlds: The Paper User Interface. In *Proceedings of ACM INTERCHI'93, Conference on Human Factors in Computing Systems*, April 1993. 215
[14] C. Kafka. DataGlyphs Bridge Paper and Digital Worlds. *Docu World*, 1998. 215
[15] A. Kobler and M. C. Norrie. OMS Java: Lessons Learned from Building a Multi-Tier Object Management Framework. In *Proceedings of OOPSLA'99, Workshop on Java and Databases: Persistence Options*, Denver, USA, January 1999. 208
[16] A. Kobler, M. C. Norrie, B. Signer, and M. Grossniklaus. OMS Java: Providing Information, Storage and Access Abstractions in an Object-Oriented Framework. In *Proceedings of OOIS'2001, 7th International Conference on Object-Oriented Information Systems*, Calgary, Canada, August 2001. 208
[17] B. M. Lange, M. A. Jones, and J. L. Meyers. Insight Lab: An Immersive Team Environment Linking Paper, Displays, and Data. In *Proceedings of ACM CHI'98, Conference on Human Factors in Computing Systems*, April 1998. 216
[18] LeapPad Learning System, LeapFrog Enterprises, Inc., http://www.leapfrog.com. 207
[19] D. M. Levy. *Scrolling Forward: Making Sense of Documents in the Digital Age*. Arcade Publishing, October 2001. 201
[20] W. E. Mackay. Augmented Reality: Linking Real and Virtual Worlds - A New Paradigm for Interacting with Computers. In *Proceedings of ACM AVI'98, Conference on Advanced Visual Interfaces*, May 1998. 215
[21] W. Newman and P. Wellner. A Desk Supporting Computer-Based Interaction with Paper Documents. In *Proceedings of ACM CHI'92, Conference on Human Factors in Computing Systems*, May 1992. 215
[22] M. C. Norrie and A. Palinginis. A Modelling Approach to the Realisation of Modular Information Spaces. In *Proceedings of CAiSE'02, 14th International Conference on Advanced Information Systems Engineering*, Toronto, Canada, May 2002. 212
[23] M. C. Norrie, A. Steiner, A. Würgler, and M. Wunderli. A Model for Classification Structures with Evolution Control. In *Proceedings of ER'96, 15th International Conference on Conceptual Modelling*, Cottbus, Germany, October 1996. 212
[24] K. O'Hara and A. Sellen. A Comparison of Reading Paper and On-Line Documents. In *Proceedings of ACM CHI'97, Conference on Human Factors in Computing Systems*, March 1997. 201
[25] OTM Technologies, VPen, http://www.otmtech.com. 215
[26] Paper^{++}, IST-2000-26130, http://www.paperplusplus.net. 201
[27] S. Pradhan, C. Brignone, J-H. Cui, A. McReynolds, and M. T. Smith. Websigns: Hyperlinking Physical Locations to the Web. *IEEE Computer*, August 2001. 216
[28] J. Rekimoto and Y. Ayatsuka. Cybercode: Designing Augmented Reality Environments with Visual Tags. In *Proceedings of DARE'2000, Designing Augmented Reality Environments*, Elsinore, Denmark, April 2000. 215
[29] A. J. Sellen and R. Harper. *The Myth of the Paperless Office*. MIT Press, November 2001. 200
[30] A. Sheth and J. Larson. Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Databases. *ACM Computing Surveys*, pages 183–236, September 1990. 202

[31] B. Signer, M. Grossniklaus, and M. C. Norrie. Java Framework for Database-Centric Web Engineering. In *Proceedings of WebE'2001, 4th Workshop on Web Engineering (in conjunction with 10th International World Wide Web Conference)*, Hong Kong, May 2001. 210, 211

[32] L. Stifelman, B. Arons, and C. Schmandt. The Audio Notebook: Paper and Pen Interaction with Structured Speech. In *Proceedings of ACM CHI'2001, Conference on Human Factors in Computing Systems*, Seattle, USA, March 2001. 216

[33] R. Want, K. P. Fishkin, A. Gujar, and B. L. Harrison. Bridging Physical and Virtual Worlds with Electronic Tags. In *Proceedings of ACM CHI'99, Conference on Human Factors in Computing Systems*, May 1999. 216

[34] P. Wellner. Interacting with Paper on the DigitalDesk. *Communications of the ACM*, 36(7), July 1993. 215

[35] G. Wiederhold. Mediators in the Architecture of Future Information Systems. *IEEE Computer*, pages 38–49, 1992. 202

Integrating Scientific Data through External, Concept-Based Annotations

Michael Gertz[1] and Kai-Uwe Sattler[2]

[1] Department of Computer Science
University of California, Davis, CA, U.S.A.
gertz@cs.ucdavis.edu
[2] Department of Computer Science
University of Magdeburg, Germany
kus@iti.cs.uni-magdeburg.de

Abstract. In several scientific application domains, such as the computational sciences, the transparent and integrated access to distributed and heterogeneous data sources is key to leveraging the knowledge and findings of researchers. Standard database integration approaches, however, are either not applicable or insufficient because of lack of local and global schema structures. In these application domains, data integration often occurs manually in that researchers collect data and categorize them using "semantic indexing", in the most simple case through local bookmarking, which leaves them without appropriate data query, sharing, and management mechanisms.
In this paper, we present a data integration technique suitable for such application domains. This technique is based on the notion of controlled data annotations, resembling the idea of associating semantic rich metadata with diverse types of data, including images and text-based documents. Using concept like structures defined by scientists, data annotations allow scientists to link such Web-accessible data at different levels of granularity to concepts. Annotated data describing instances of such concepts then provide for sophisticated query schemes that researchers can employ to query the distributed data in an integrated and transparent fashion. We present our data annotation framework in the context of the Neurosciences where researchers employ concepts and annotations to integrate and query diverse types of data managed and distributed among individual research groups.

1 Introduction

The past few years have witnessed a steady improvement of standard database integration techniques to deal and scale with data on the Web [12, 2]. Data wrapper and mediator techniques are just a few of such approaches that basically investigate the schemas of Web-accessible data sources and either materialize (wrapped) data into a globally accessible data warehouse or provide for global queries mediated among the sources.

S. Bressan et al. (Eds.): EEXTT and DIWeb 2002, LNCS 2590, pp. 220–240, 2003.

While these approaches are suitable for data sources that exhibit database like structures, they are not applicable or suited for integrating data from sources where such schema-like structures are almost non-existent. The most prominent example of such scenarios are the computational sciences, such as computational biology, Neurosciences, astrophysics or medicine. In these domains, the data typically range from database like sources over Web documents to high-resolution images. For example, while an image may exhibit the same information content as a few rows of a relation or a paragraph in a text document, it is almost impossible to automatically integrate and represent these "data" or rather regions of interest in one globally accessible structure. The typical way in which users integrate such data is that they categorize them using their domain knowledge and expertise and then build structures that either cluster the data in a materialized fashion or simply keep link structures to the Web data.

In this paper, we present a coherent framework that allows scientist to utilize *concepts*, representing well-defined, agreed upon domain specific features of interest, to semantically enrich Web-accessible data and to link such data to concepts as instances of such concepts. While concepts and relationships among concepts then can be considered as a global schema, the data, which can be regions of interest (ROIs) in a document and are further described by scientists in terms of their properties, represent instances of such concepts. Relationships between concepts can be inherited to relationships among data and thus can be considered as typed (Web-)links among data that are external from the data. In other terms, annotations represent metadata that follow the structures specified by concepts. Concepts, annotations and Web data then build a graph structure for which we present a flexible query language and its realization as basis for different services that can be build on annotations, such as a search engine or concept browser.

The following gives an example from a project currently conducted at the Center for Neuroscience at UC Davis in the context of the Human Brain Project [17, 22, 31]. In this project, Neuroscientists at different sites generate various types of data related to the human brain, including high resolution images of anatomical structures, bio-physiological or chemical properties of cells, nuclei, and their connections in form of graphs or charts, and text data describing findings regarding specific features of interest, such as the processing of information by cell structures. With none of theses data a schema is associated but they are all accessible through the Web where, e.g., centralized image or chart registries are employed to publish data.

Assume a researcher browsing a newly generated set of images published by a research group (left browser window in Figure 1). In one of these images, the researcher identifies a region containing a cell structure of a known type A. Ideally, she would like to annotate this region and "link" it to a conceptual structure that has been defined in the specific research context. This structure defines a concept (similar to a metadata schema) that details general, agreed upon properties of the cell type A such as a definition, terms typically used to refer to the cell type, and other general properties of that concept. Through this linkage, the identified

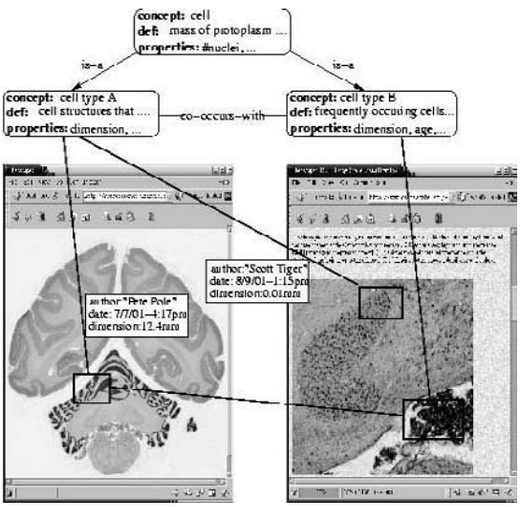


Fig. 1. Scenario of Concept-based Annotations for Scientific Data

region in the image then can be understood as an *instance* of the concept and which then is filled out by the researcher (e.g., specific values for that instance in this image). Assume the researcher knows about another group that deals with similar images, generated in the context of a different experiment. She pulls up one such Web document which contains some images as well as some descriptive text (right browser window). The image in this document exhibits a region containing the cell type A. Again, she would like to link this region to the same concept used for the first image, thus specifying another instance of this concept with perhaps different values for the properties. The aspect of linking (annotating) these regions to a concept and instantiating concept properties is indicated through the lines and boxes at the marked regions in Figure 1. Linking different regions of interest, independent of whether these regions are within an image or text document then corresponds to a data integration process where the researcher manually classifies data and associates them with respective structures. It should be noted that none of the aspects described in the above scenarios can be accomplished by applying known integration methods for Web data, unless sophisticated feature detection mechanisms can be provided for the automatic discovering of concept instances.

In the following, we present the conceptual framework that supports the modeling and querying of scenarios such as the above. The annotation graph model described in Section 2 provides expressive means to specify diverse types of relationships among concepts, documents, and regions of interest, and it also provides query mechanisms to traverse such graph structures representing inte-

grated data. In Section 3, we then outline the integration process for scientific data in the context of the annotation graph model. Section 4 discusses the realization of an integration platform implementing the model and services based thereon. After a presentation of related work in Section 5, we conclude the paper in Section 6 with a summary and some future research.

2 Integration Model

Integrating data from different sources requires correspondences among schemas to some extent. Typically, the sources to be integrated offer some kind of schema, which represents a sub-/superset of a global schema. However, application domains as those described in the introduction in general lack such kind of data source schemas. Instead, correspondences among structures and data exist on a semantic level only where regions of interest in documents can be understood as instances (data) of certain concepts (schemas). In this respect, integrating documents and the data they represent means to enable queries on automatically extracted or manually assigned features of the documents. Such features can be represented through annotations. An annotation of a document basically can be understood as a typed link between an object (so-called *region of interest* or *ROI*) in the document and a domain specific concept representing a metadata template. Associated with such an annotation are property values that describe the feature according to the concept. In order to specify, represent, and in particular query concepts, annotations and documents in a uniform fashion, these types of information need to be modeled appropriately and a respective model should be easy to implement and use in different data management and retrieval tasks.

The model should be extensible with respect to different conceptual structures, such as Dublin Core [7] in the most simple case, Neuronames [21] or UMLS [5] as examples for specific Neuroscience or medical vocabularies, as well as complex ontology-like structures. Such conceptual structures are typically developed in a collaborative fashion and represent various domain specific aspects. Note that conceptual models are not the primary focus of our work. They are mainly used to provide common semantic "anchors" for annotations associated with documents. Thus, we rely on work in the context of knowledge models as developed, e.g., by the Semantic Web community. In the following, we detail our annotation graph model that is based on a simple conceptual structure underlying our approach to annotate regions of interest in documents for data integration purposes.

2.1 Annotation Graph Model

The annotation graph model represents concepts, annotations and (regions of) documents as nodes in a graph and they are linked by typed relationships. The model serves as an integration model defining connections among a global metadata schema and distributed and heterogeneous document collections.

Assume a set $T = \{String, Int, Date, \dots\}$ describing a set of simple data types. Let $\mathrm{dom}(T)$ denote the set of all possible values for T. In the annotation graph model, a property of a node is defined as an identifier and a type $PDef = String \times T$. An instantiation of a property consists of an identifier and a value $PVal = String \times \mathrm{dom}(T)$.

As outlined above, concepts provide templates for annotations that are associated with documents. In our model, concept are represented by a simple yet extensible form of *concept nodes*. We assume that each concept node has (1) a concept identifier (typically the preferred name for the concept), (2) some descriptive terms or phrases typically used for the concept in data retrieval tasks ($\mathbb{P}\ Term$, $Term \subseteq String$), (3) a natural language definition ($Def \subseteq String$) that associates an agreed upon, well-defined meaning with the concept, and (4) a set of concept property specifications $\mathbb{P}\ PDef$. In this respect, a concept definition is similar to a class definition in the context of object-oriented modeling. In the following, we denote the set of all concepts by $\mathcal{C}$, where $\mathcal{C} \subseteq Term \times Def \times \mathbb{P}\ Term \times \mathbb{P}\ PDef$.

The second type of node in our graph model represents Web accessible documents. A document is identified by its URI (Uniform Resource Identifier) of type *String* and optionally has a set of document properties such as author, title, etc. We denote the set of all Web documents by $\mathcal{D}$.

Finally, annotation nodes provide the basis for specifying links between concepts and documents. The set of annotations $\mathcal{A}$ is defined as $\mathcal{A} \subseteq String \times Date \times \mathbb{P}\ PVal$. For a tuple $(creator, created, pvals) \in \mathcal{A}$, *creator* is the author of the annotation, *created* is the creation date/time, *pvals* is a set property instantiations. The set of all nodes $\mathcal{V}$ in an annotation graph is now defined as $\mathcal{V} = \mathcal{A} \cup \mathcal{C} \cup \mathcal{D}$. Links between nodes are represented as directed, typed edges, to which property instantiations are optionally assigned. The types of edges are drawn from the specified concepts (see below) and the property instantiations are determined by the associated concept. Finally, the set $\mathcal{E}$ of all edges is define as

$$\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V} \times \mathcal{C} \times \mathbb{P}\ PVal$$

The meaning of the components of an edge $(from, to, type, pvals) \in \mathcal{E}$ is as follows: the edge connects the node *from* with the node *to* (in that direction). With the edge the concept *type* is associated, and *pvals* is a set property instantiations.

For example, to represent an "is-a" relationship between two concepts c_{super} and c_{sub} we define an edge $(c_{sub}, c_{super}, isA, \emptyset)$ where *isA* is the identifier of a *relationship type concept*. In an implementation of this model, the kind of nodes connected by an edge should be further restricted, but this is not addressed in this basic model.

Using these definitions, our annotation graph model comprises both the metadata components (concepts) and data components (annotations and documents). An instance of the model then is represented by a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$. Naturally, nodes can be connected via edges of arbitrary types. However, in most cases only edges of certain types make sense. In order to deal with the special meaning of the different kinds of nodes and how they can be connected, we introduce the

following *default relationship type concepts*. They are contained in any specification of a collection of concepts and can be extended by more relationship type concepts to relate concepts to each other in a well-defined way.

- *annotates* is used to represent edges from annotations to documents. If an annotation is specified for a document, a link referring to this relationship type concept is defined between both of them. Since we are interested in fine-grained annotations, e.g., regions in an image or fragments of a text document, we assume a set of *document context descriptions*. A description comprises information about the document context in which an annotation is specified and is modeled as a property of the relationship. In our current system, we employ two types of descriptions. For text-based documents, we employ a tree-structured document model. In particular, we use a transformation mechanism to handle HTML and plain text documents as XML documents. XPath expressions [4] are then used as document context descriptors. For fine-grained annotations in image data, we use a spatial region object, i.e., a sequence of 2D points representing a polygon, and scale information to encode respective context descriptions.
- *annotatedBy* is a relationship type concept representing the inverse of *annotates*.
- the concept *ofConcept* specifies that an annotation is based on a certain concept, i.e., it assigns the concept to the annotated document and instantiates the properties defined by this concept.
- *hasAnnotation* describes the inverse of the relationship *ofConcept*. In addition, each annotation $a \in \mathcal{A}$ must be related to a concept.
- *isA* denotes an "is-a" relationship type concept between concepts, thus implementing inheritance of property definitions, i.e., $\forall e \in \mathcal{E} : e.rel = isA \rightarrow e.from, e.to \in \mathcal{C} \wedge e.from.pdefs \supseteq e.to.pdefs$. As mentioned above, other relationship type concepts can be introduced to model how (base) concepts are related. This can include spatial, temporal, and general semantic types of relationships, as outlined in [32].

Figure 2 illustrates the usage of these modeling primitives in an annotation graph. The concept level comprises the application-specific concepts "cell" (C2), "cell type A" (C3) etc. as well as the relationship type concept "is_of_cell_type" (C1) and the builtin concepts for representing relationships such as "ofConcept" (C7), "annotates" (C5) etc.

At the annotation level the annotations as instances of the cell concepts are used for linking the document to the concepts. The context descriptors are properties of the relationship "annotates" and specify the relevant fragment or region of the document. In this way, a single annotation object can be associated with more than one document simply by establishing multiple relationships between the annotation and the documents. This is particularly useful for bulk annotations. In principle, a single annotation could be an instance of more than one concept, but this is forbidden at the system level. Naturally, a document can have many annotations and a concept can be used to annotate several documents (or regions thereof).

Though we have discussed our model so far in our own notation it should be noted that this model can be easily mapped to RDF or RDF Schema (RDF/S) respectively. Here, concepts are modeled as RDF/S classes with associated properties and both annotations and documents are represented as resources. The edges of our graph model have to be mapped to RDF/S classes with properties which represent the links to the connected nodes as well as the property instantiations. So, we could interpret our graph model as a vocabulary specified in RDF/S. However, in order to simplify the view for querying and navigating in annotation graphs, in the following we use the graph model and describe enriched query operations.

2.2 Query Operations

In order to effectively utilize a collection of annotated documents, certain query operations need to be supported. For a graph-like structure as adopted for our annotation graph model, such operations basically consist of node selection and path traversal. The input for a selection is always a homogeneous set: either one of the basic sets $\mathcal{A}$, $\mathcal{C}$ or $\mathcal{D}$ (but not a union of them) or a derived set resulting from a prior operation. Let S be one of the sets $\mathcal{A}$, $\mathcal{C}$ or $\mathcal{D}$ and $P(s)$ a predicate on $s \in S$. Then the selection operation Σ_P is defined as

$$\Sigma_P(S) = \{s \mid s \in S \wedge P(s)\}$$

A predicate P is a boolean expression made up of clauses of the form *prop* <*op*> *value* and which can be combined through logical connectives. In addition, path

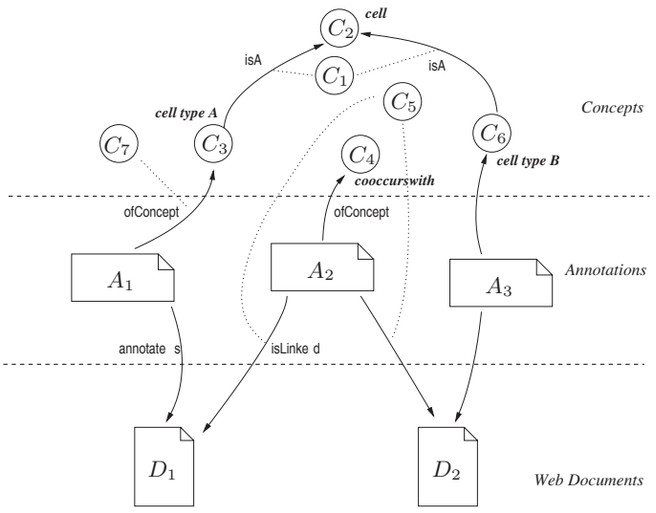


Fig. 2. Example of an annotation graph

expressions of the form $rel_1.rel_2 \ldots rel_n.prop$ specifying the traversal of edges (relationship type concepts) rel_1, rel_2 etc. from the current node to the property *prop* of the target node are allowed as long the result is a single-valued expression. Path traversal enables following links between nodes of the graph. Given a start node v_s and a relationship type concept rel, the operation ϕ_{rel} returns the set of target nodes based on existing edges:

$$\phi_{rel}(v_s) = \{v_t \mid (v_s, v_t, rel) \in \mathcal{E}\}$$

Since we mainly have to deal with sets of nodes in query expressions, this operation is also defined on a set of nodes $V \in \mathcal{V}$:

$$\Phi_{rel}(V) = \{v_t \mid \exists v_s \in V : (v_s, v_t, rel) \in \mathcal{E}\}$$

A special kind of the path traversal operation is the operation for computing the transitive closure. This operator extends ϕ_{rel} by traversing the path indicated by the relationship *rel* as long as edges can be found that have not already been visited. The set of all nodes traversed thus is defined as

$$\phi^+_{rel}(v_s) = \{v_t \mid (v_s, v_t, rel) \in \mathcal{E} \vee \exists v_i \in \phi^+_{rel}(v_s) : (v_i, v_t, rel) \in \mathcal{E}\}$$

As for ϕ_{rel}, this operation is defined on a set of nodes, too:

$$\Phi^+_{rel}(V) = \{v_t \mid \exists v_s \in V : (v_s, v_t, rel) \in \mathcal{E} \vee \exists v_i \in \Phi^+_{rel}(v_s) : (v_i, v_t, rel) \in \mathcal{E}\}$$

Using these basic operations, query expressions for selecting nodes and traversing edges can be formulated. Note that the initial set of nodes for a traversal always has to be obtained by applying a selection on one of the basic sets $\mathcal{A}$, $\mathcal{C}$ or $\mathcal{D}$. From this, following the relationships type concepts *ofConcept*, *annotates* etc. allows to go to another type of nodes.

For the purpose of more developer-friendly query formulation, we have designed a simple language in the spirit of XPath. Here, the sets $\mathcal{A}$ (*annotations*), $\mathcal{C}$ (*concepts*), and $\mathcal{D}$ (*documents*) are valid root elements. If views on these sets are defined, they can be used as root elements as well. Selections are formulated by appending `[`*condition*`]` to a term. In *condition*, the properties of nodes can be specified and —in combination with the usual operators and logical connectors— used for formulating complex predicates. The Φ-operator is expressed by appending `/`*relship* to the term. *relship* denotes the relationship type concept that has to be used for following the links. The optional `+` indicates that the transitive closure has to be computed.

The following is a simple example of a query that specifies all documents annotated using a given concept named *C*. First, the desired concept is selected. Then, by traversing the relationships to the annotations (*hasAnnotation*) and then to documents (*annotates*), the final result, here a set of documents, is obtained:

$$\Phi_{annotates}(\Phi_{hasAnnotation}(\Sigma_{\text{name='C'}}(\mathcal{C})))$$

An equivalent query, written in our query language, looks as follows:

concept[name='C']/hasAnnotation/annotates

Additional selections can be applied to each of the intermediate results of the query. As an example, it is possible to restrict the annotations to be considered to a certain author or date. If all those documents are to be retrieved that are either annotated using a concept C or a concept that is a (in)direct subconcept of C (assuming the relationship type concept *subtype* is the inverse of *isA*), the above query is extended to

concept[name='C']/subtype+/hasAnnotation/annotates

After selecting the concept named C, all concepts in its transitive closure with respect to the *subtype* relationship are obtained. From this set, the annotated documents are determined as described above. While these examples illustrate the querying of distributed documents in a uniform fashion using concepts, the next example shows how "related" documents can be determined:

document[uri='www.... ']/annotatedBy/ofConcept/hasAnnotation/annotates

Starting from a document with a given URI, first all annotations for that document are determined. Following the *ofConcept* relationship, their underlying concepts are obtained, and by going "downwards" from there to the annotations and documents, we can find "related" documents (or regions), i.e., documents that have been annotated using the same concepts as the specified document.

Since annotations are first-class objects, they can be linked to more than one document. That is, they can be used for annotating several documents at the same time. In the following, this aspect is utilized by retrieving all documents that are linked to a given document by an annotation based on concept C:

document[uri=...]/annotatedBy[ofConcept.cname='C']/annotates

In summary, the proposed annotation graph model and associated query language supports a wide variety of query operations against distributed collections of annotated documents, the most important classes of queries being

- *concept* → *document*: Starting from a concept (at the schema level), documents annotated using that concept are retrieved. This operation is equivalent to retrieving the extension of a given class.
- *document* → *concept* → *document*: Documents annotated by the same or a similar concepts are retrieved. This corresponds to a concept-based similarity operation.
- *document* → *document*: In this class of queries, links represented by annotations are utilized for finding related documents. This operation is similar to a join between data objects.

2.3 Views

In collaborative research environments such as those described in the introduction, data in form of documents are continuously generated by researchers at

different sites. In order to allow for different foci of interest in an integrated collection of documents, views are an essential mechanism for structuring or grouping concepts (perhaps originating from different existing vocabularies), annotations, and documents according to such foci and research interests. In particular, views allow to support different vocabularies or conceptual structures, as they frequently occur even among research groups that have the same research interest.

In our annotation graph model, a view restricts the set of nodes to be considered in queries. More precisely, a view specifies a (virtual) sub-graph $\mathcal{G}' = (\mathcal{V}', \mathcal{E}')$ of an annotation graph $\mathcal{G}$. Due to the distinction of several node types, $\mathcal{V}'$ is specified through separate queries on the base sets $\mathcal{A}$, $\mathcal{C}$, and $\mathcal{D}$ using query operations presented in Section 2.2. For example, if we want to provide a view containing only (1) concepts that originate from the Neuronames collection, and (2) annotations made this current year, we could define this as follows:

```
define view my_view as
        annotation := annotation[created >= '01/01/2002']
        concept    := concept[origin = 'Neuronames']
```

If a base set is not restricted in the view definition, as in the above case the document set, the complete set is used when querying the view. For individual restricting queries, any valid query expression is allowed as long as it returns a proper subset, e.g., a set $C' \subset C$ of the concept set. Because the set of concepts contains essential concepts like *isA*, *annotates* etc., the set of default relationship type concepts is treated in a special way, guaranteeing the inclusion of the built-in concepts in each view.

It should be noted that we currently do not provide a separate view definition language. Instead a view is defined using a dedicated service. A view is used in a query by simply giving its name as an additional parameter to the invocation of the query processor. The view resolution itself is performed as part of the query translation, which is detailed in Section 4.

3 Integration Process

The primary task in integrating scientific data (or documents) using the annotation graph model is to create concepts and relationships among them, and to establish mappings between regions of interest in documents and concepts using annotations. In this context, two major issues arise:

- The mapping cannot be done automatically because it requires an interpretation of the document content. For example, a certain region in an image has to be identified by a researcher as an image of an instance of the concept "cell". While it is conceivable that for text documents such a conceptualization can be done in a semiautomatic fashion, using, e.g., NLP techniques, thesauri etc., in the general case and for images in particular one has to rely on the manual approach.

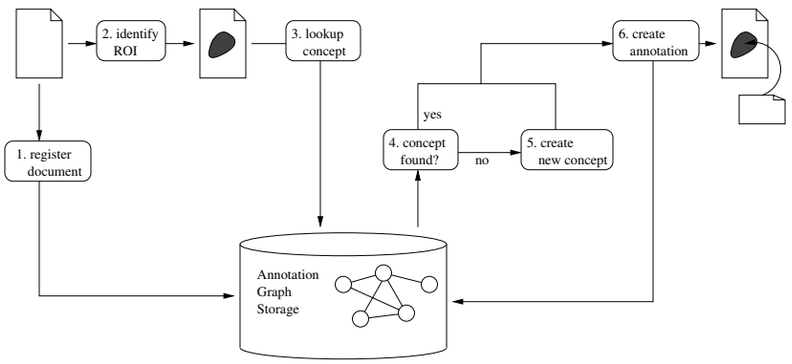


Fig. 3. Process of integrating schema-less data using concepts

– How does one define concepts representing a form of global metadata schema? In principle, for a given application domain, a schema consisting of concepts and relationships could be modeled in advance and used for annotating distributed documents. However, often not all domain specific concepts are known in advance but are introduced while researchers investigate the information content of documents. This is particular true if new information is discovered that cannot be associated with any existing concept. In this respect, the concept part of an annotation graph can be a dynamic and growing structure, and the integration process has to take this aspect of an "evolving schema" into account.

Due to these characteristics, we propose the following process for integrating data through concept-based annotations (Figure 3):

1. An initial set of agreed-upon concepts is specified, including the most important and obvious relationships types concepts such as sub/supertype or, as it is the case in for neuroanatomical data about the human brain, spatial relationships, and stored in a centralized annotation graph storage. For example in the Human Brain Project, where information about the visual pathway in the brain are to be integrated, one could start with a set of concepts organized in a containment hierarchy such as *brain* → *cerebrum* → *diencephalon* → ...
2. For annotating a document, that is, integrating the document and its information content into the global collection, an appropriate concept has to be chosen. Based on this concept, an annotation is created and the relevant region in the document is marked, e.g. in the above mentioned scenario by drawing a polygon in the image of the brain slice. Futhermore, the properties defined by the concept have to be instantiated. This means, the researcher has to provide values such as the diameter of the structure or other describ-

ing information. If no appropriate concept exists, a new concept is specified (perhaps by deriving it from an existing concept) and added to the concept collection. In addition to this, relationships to other concepts can be specified. That means, if a new structure in a brain image was discovered, one would create a new concept, add it to the appropriate concept as a sub-part, and create an annotation as instance of this concept.

Of particular interest in data integration scenarios is also the handling of structural and data conflicts. In [14], we have detailed an approach to deal with such scenarios in the context of annotating image data. Here, we will give only a brief overview of the basic ideas.

An annotation graph typically is constructed in a collaborative fashion. That is, several users introduce and modify concepts and annotations over time. In order for the annotations to be useful in data retrieval, services need to be provided that check for a certain degree of consistency in terms of how concepts and annotations have been used. There has been quite a lot of work in the context of the creating and sharing ontology like structures, e.g., [13, 29, 10], and it is well known that consistency issues in such structures pose a hard and in general not completely solvable problem.

The basic idea we employ to deal with such scenarios is that if concepts, as core components of such ontological structures, are used to annotate data, such annotations can provide important input to some consistency aspects. Consider the scenario where some agreed upon base concepts have been introduced and these concepts are used by different users in annotating documents. Different users might have a different focus and thus it is totally reasonable that, e.g., different concepts are used to annotate the same documents and regions of interest. This is a natural aspect in collaborative research environments, in particular if features are detected in data for which no concepts have been introduced yet.

In our current system prototype, we use information about how annotations are used to provide users with some feedback regarding the creation of annotations and concept. This feedback then can be used to investigate possible inconsistencies (incompatibilities) by, e.g., a group of domain experts who review submitted concepts or annotations. Among others, the following important scenarios are considered:

Annotation granularity and redundant concepts. Assume a user specifies an annotation A_1 for a document based on a concept C_1. With each annotation, a context is associated that details the granularity of the annotation. This can be either a spatial object (e.g., polygon) in case of an image, or an XPath expression in case of a text document (see also Section 2.1). During annotation creation time, the system checks whether there already exist annotations A' for the same document such that there is a containment relationship between annotation contexts (which are well-defined for spatial objects and subtree structures). Assume there is another annotation A_2, based on concept C_2 such that the context for A_2 is contained in the context for A_1. If both annotations refer to the same concept, then for either A_1 or A_2 there should be a modification. If A_1 and A_2 use the same context but different concepts, then this either indicates an inconsistency

or just a different viewpoint among the users. In particular the first aspect turns out to be very valuable since users get an insight into how other users interpret the data in a perhaps different research context. Other scenarios based on containment of document contexts and relationships (in particular "is-a" relationships) among used concepts are checked by the system and are discussed in more detail in [14].

Propagating modifications. The deletion of a concept naturally implies the deletion of all annotations that are based on this concept. However, in case an "is-a" relationship exists between the deleted concept and some other concept, another option would be to "relink" the annotations either to the more specific or more abstract concept, depending on the direction of the "is-a" relationship. Such a scenario can take place in a user-guided fashion and preserves annotations and concept instance properties to a certain degree. Finally, the deletion of a document results in the deletion of all annotations associated with that document.

In summary, document and data integration as proposed in this paper is an iterative, evolving process with continuous modifications at the concept (schema) and annotation level. We support this by providing tools for querying and browsing concepts and by implementing mechanisms for compatibility and integrity checks.

4 Implementation of the Integration Platform

In this section, we describe the implementation of the presented framework and its application in our project conducted in the context of the Human Brain Project. In this project, brain slices are digitally photographed under a microscope and utilized by researchers who identify and mark specific regions (e.g., individual cells or cell structures) and assign concepts (e.g., a cell type) to these regions. The annotation graph model is used to represent content descriptive metadata templates in form of concepts as well as relationships among such concepts. In this way, it provides an integrated view on the annotated features of documents distributed among Web sources. Furthermore, the query model allows to formulate declarative queries for traversing the graph and retrieving data.

The integration platform is realized by an annotation server, which provides basic annotation and query services. The architecture of this server is shown in Figure 4. All components of the system are implemented in Java using JDBC for accessing the DBMS. The interface to the services of the system is realized using Apache SOAP. In detail, the following services are provided:

- creating, modifying, and deleting concepts,
- registering and unregistering documents,
- creating, modifying and deleting annotations as instances of concepts,
- creating and dropping views,
- loading an entire annotation graph from a given XML document,
- querying the annotation graph.

Each of these services is implemented as a SOAP RPC service and thus can be invoked from other clients which need to create or retrieve concepts and annotations. For example, the annotation tool is a Java-based client software using these services for annotating images of brain slices.

The main component of the system is the graph mapping module. It realizes the mapping of the conceptual model of an annotation graph consisting of nodes and edges to relations stored in a relational database. The database has the following relation schemas:

CONCEPT*(id, def)*
CONCEPTTERM*(id* → CONCEPT*, term)*
CONCEPTPDEF*(id* → CONCEPT*, kind, name)*
CONCEPTRELSHIP*(id*→ CONCEPT*, fromTable, toTable)*
DOCUMENT *(id, uri, title)*
ANNOTATION*(id, creator, created)*
ANNOTATIONPVAL *(id*→ ANNOTATION*, name, val)*
RELSHIP *(id, from, to, rel*→ CONCEPT*)*
RELSHIPPVAL *(id*→ RELSHIP*, name, val)*

Concepts are stored in the normalized relations CONCEPT, CONCEPTTERM (for terms), and CONCEPTPDEF (for property definitions). If a concept is a relationship type concept, an additional tuple is created in the relation CONCEPTRELSHIP specifying which kind of nodes (identified by their names) are linked. Information about documents is managed in the DOCUMENT relation, and an-

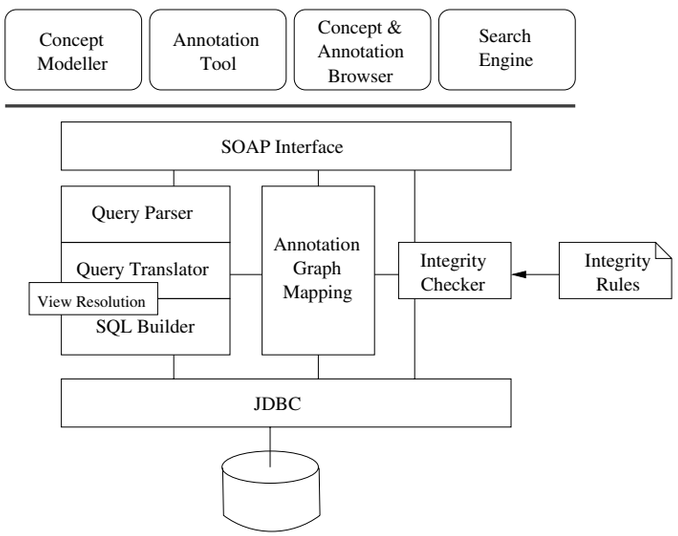


Fig. 4. Architecture of the annotation server

notations are stored in the relation ANNOTATION together with the property values in ANNOTATIONPVAL. Finally, the relations RELSHIP and RELSHIPPVAL are used to manage relationships type concepts and properties of respective instantiations.

The main advantages of such a generic mapping approach are that (1) new concepts with their properties can be easily defined without any schema modifications or extensions and (2) the query translation module can be implemented without knowledge of the concrete concepts and their mappings. The disadvantage is the complexity of the translated SQL queries which consist of many joins. This could be avoided by using a mapping scheme optimized for the given concept structures, i.e. by grouping concepts and properties together. However, especially in integration scenarios modifications or additions to the concept level are very likely, for example as a result of adding a new source. So, a generic mapping scheme provides more flexibility.

The query model is tightly coupled with the mapping module. It consists of the parser for the query language, a translator that transforms a given query into a relational algebra expressions, and an SQL builder for deriving and executing the corresponding SQL query. The transformation from a query expression into relation algebra is specified by a set of rules. We assume the well-known relational algebra operators: σ_{cond}, $\bowtie$, $\bowtie_\theta$, π, ρ (for renaming) as well as the recursive operator α [1]. Utilizing the α operator is a feasible approach since most of today's database systems support some kind of recursive queries, e.g., Oracle with the `CONNECT BY` clause or IBM DB2 with `RECURSIVE UNION`, or allow at least to implement user-defined table functions that can perform transitive closure computation. We use the notation

$$\omega(E) \mapsto \varpi(E')$$

to express the transformation of an expression $\omega(E)$ into a relational expression $\varpi(E')$ by substituting ω with ϖ and applying it to the rewriting of E which is denoted by E'. Furthermore, *eval*(E) denotes the evaluation of the expression E, and *relation*(T) denotes the relation with name T.

The transformation of a query expression into relational algebra can now be specified by the following set of rules:

(1) $\mathcal{A} \mapsto$ ANNOTATION
(2) $\mathcal{C} \mapsto$ CONCEPT
(3) $\mathcal{D} \mapsto$ DOCUMENT
(4) if *eval*$(E) \subseteq \mathcal{A}$ then
$\Sigma_P(E) \mapsto$
$\sigma_{\overline{P}}(E') \bowtie \rho_{T_1}(\text{ANNOTATIONPVAL}) \bowtie \cdots \bowtie \rho_{T_n}(\text{ANNOTATIONPVAL})$
Here, $\overline{P}$ is constructed as follows. For each clause c_i of the form "*prop op val*", there is a corresponding clause "T_i.name = '*prop*' $\wedge$ T_i.val *op val*".
(5) if *eval*$(E) \subseteq \mathcal{D}$ then $\Sigma_P(E) \mapsto \sigma_P(E')$
(6) if *eval*$(E) \subseteq \mathcal{C}$ then $\Sigma_P(E) \mapsto \sigma_P(E' \bowtie \text{CONCEPTTERM})$
(7) $\Phi_r(E) \mapsto T_{to} \leftarrow \pi_{to}(\sigma_{id=r}(\text{CONCEPTRELSHIP})$;
$\pi_{T_{to}.*}(E' \bowtie_{id=from} (\sigma_{rel=r}(\text{RELSHIP})) \bowtie_{to=id} \mathit{relation}(T_{to}))$

(8) $\Phi_r^+(E) \mapsto T_{to} \leftarrow \pi_{to}(\sigma_{id=r}(\text{ConceptRelship}));$
$\pi_{T_{to}.*}(E' \bowtie_{id=from} \alpha(\pi_{from,to}(\sigma_{rel=r}(\text{Relship})) \bowtie_{to=id} \mathit{relation}(T_{to})))$

Rules (1)–(3) specify simple substitutions of sets by the corresponding relations. In rules (4)–(6), the selection operator is translated utilizing the normalization of the relations, e.g. in rule (4) by constructing a join with the AnnotationPval table for each annotation property in the selection condition. Rule (7) translates the path traversal operator into a 3-way join between the relations *from*, *to*, and Relship. Here, the name of the *to* relation is first obtained from relation ConceptRelship which returns either Annotation, Concept or Document, depending on the kind of the relationship. The transformation of path expressions as part of a selection condition is handled in a very similar way and therefore omitted here. Finally, using rule (8), the path traversal operation based on the transitive closure is translated into an algebra expression that computes all edges of the given relationship type concept and joins them with the *from* and *to* attributes using the α operator.

These transformation rules are always applied in an inside-out manner. For example, for the query expression $\Sigma_{author='Ted'}(\Phi_{hasAnnotation}(\Sigma_{name='C'}(\mathcal{C})))$, the resulting algebra expression is

$$\sigma_{author='Ted'}(\pi_{\text{Annotation}.*}(\sigma_{name='C'}(\text{Concept} \bowtie \text{ConceptTerm}) \bowtie_{id=from} \sigma_{rel=hasAnnotation}(\text{Relship}) \bowtie_{to=id} \text{Annotation} \bowtie \text{AnnotationPval}))$$

From this algebra expression, an equivalent SQL query can easily be derived. Only the α operator requires a special treatment. Due to the limitations in the version of the Oracle DBMS used in our implementation, we have implemented this operator by a table function *tclosure (src_id, rel)*. This function computes the transitive closure of a node identified by *src_id* with regard to the relationship *rel* on the fly and returns a table of node identifiers. In this way, a query like

concept[name='C']/subtype+

is translated into the following SQL query

```
select   ct3.*, t4.*, p5.*
from     Concept c1, ConceptTerm t2, table(tclosure(c1.id, "subtype"))
         as ct3,
         ConceptTerm t4, ConceptPdef p5
where    c1.name='C' and c1.id=t2.id and ct3.id=t4.id and ct3.id=p5.id
```

The primary application for utilizing the integrated documents and data is a search engine (Figure 5). It evaluates queries formulated in our query language using the query engine of the annotation service and displays the results. Depending on the type of results (annotations, concepts, or documents (regions)) it additionally displays links to the associated objects, i.e,

– for documents (in our application domain these are images), links to the annotations,

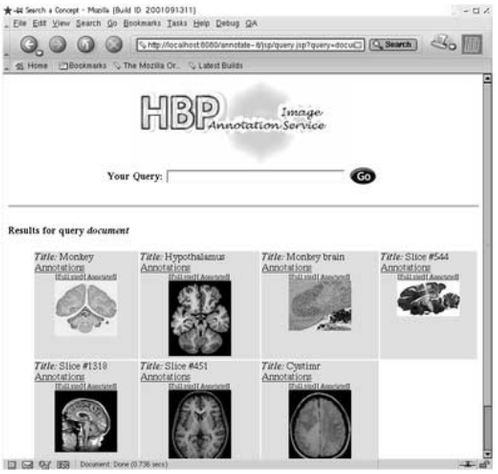


Fig. 5. The search engine

- for an annotations, the annotated documents as well as the concepts and instance properties underlying the annotations, and
- for a concept, all annotations based on this concept.

All these links are represented as queries. In this way, both querying and browsing are supported as paradigms for interacting with the integrated data. Furthermore, since a query result always represents a homogeneous set of nodes, a query can be refined. That is, a new query can be applied to the currently shown result set.

The integration of document data is done by researchers using specific annotation tools. In our application scenario, this is a tool for annotating images. This tool allows users to load and display high-resolution images where regions of interest are marked and annotated using the template induced by a previously selected concept. This concept is selected using a browser for querying and traversing the concept part of an annotation graph. If no concept satisfying the selection criteria can be found, e.g., in case of newly discovered structures, a new base concept can be defined. existing one. Creating an annotation requires not only to specify a region of interest and select an appropriate concept, but also to instantiate properties defined by the concept. These property values describe the concrete instance represented by the region and can be utilized later in queries.

5 Related Work

Researchers in computational sciences are now facing the problem of sharing, exchanging, and querying highly heterogeneous forms of data generated by ex-

periments and observations in an uniform and transparent fashion. However, because of the heterogeneity of the data and the lack of common schemas, standard approaches to data integration, e.g., based on multidatabases or federated databases, are often inapplicable or not efficient. Thus, several new approaches focusing on semantic interoperability have been proposed in the past few years.

First, there is an increasing amount of work on models and methodologies to semantically enrich the Web (see `www.semanticweb.org` for an extensive overview). The major focus in these works is on building semantic rich and expressive ontology models that allow users to specify domain knowledge. The most prominent approaches in this area are the Ontobroker project [8, 9], SHOE [15], the Topic Maps standard [30] as general ontology frameworks, and TAMBIS [26] and OIL [27] as specific ontology frameworks tailored to the biological domain. We consider these ontology-centric works as orthogonal to our annotation-centric approach. Furthermore, whereas the above approaches concentrate on querying ontologies using, e.g., RDF-based languages, our focus is to have a simple, expressive, and easy to implement language that allows to query all three components, concepts, annotations, and Web accessible documents in a uniform fashion.

A second class of approaches in this context focuses on information integration by combining query or mediator systems and domain models/ontologies. For example, in the OBSERVER system [20], metadata and ontologies represent knowledge about the vocabularies used in the sources and are utilized to handle heterogeneity for query processing. In the SCOPE system [23], semantic relationships between schema elements of different sources are identified using ontologies and exploited for evaluating queries. The MOMIS approach [3] is based on a shared ontology for defining an integrated view on heterogeneous sources and semantic optimization of queries. The mediator described in [19] uses domain maps representing semantic nets of concepts and relationships and semantic indexes of source data into a domain map. Recent surveys on current works in this area are given, e.g., in [28] and [24]. However, all these approaches require database-like structures and schema information. Thus, they are not applicable to schema-less data such as images or textual documents.

At the other end of the spectrum, several systems have been proposed that provide users with means to annotate data. This includes the multivalent document approach [25], the SLIMPad approach [6], the Annotea project [18] as well as some commercial systems (see, e.g., [11, 16] for an overview). While none of these approaches supports a query framework for annotations, only [6] support the notion of concept like structures underlying annotations.

6 Conclusions and Future Work

In many computational sciences, the association of different types of metadata with heterogeneous and distributed collections of scientific data play a crucial role in order to facilitate data retrieval tasks in an integrated and uniform way. In this paper, we have presented a framework that allows researchers to associate

well-defined metadata in form of concept instances with images and text data. This framework comprises a graph-based data model, operations for traversing graphs of concepts and annotations as well as a view mechanism. The model and its realization provide all features researchers in collaborative environments deem necessary to enrich data and thus to "semantically" index and integrate data that is not easy to classify and analyze otherwise.

Furthermore, we have described the mapping of our model to a relational database schema and an appropriate query translation scheme. In this way, we can rely on efficient data management facilities provided by full-featured DBMS without exposing the user to the complexity of pure SQL queries for data retrieval in the graph structure.

While the usage of the first prototype of our system has shown that the generic mapping approach achieves a reasonable performance for moderate-sized graphs, we are currently investigating more sophisticated mappings, e.g., involving materialization of paths for frequently used relationships such as the relationships of the "built-in" types.

Further challenges are the scalability of the system as well as effectiveness of user interfaces. The scalability issue addresses the support of larger communities. A distributed approach using multiple instances of the graph storage and query services could provide this but requires replicas of essential metadata like concepts and partitioning/clustering of annotations and relationships. With respect to the user interface we plan to investigate visual query interfaces exploring the graph structure of the data.

References

[1] R. Agrawal: Alpha: An Extension of Relational Algebra to Express a Class of Recursive Queries. In *Proc. 1987 ACM SIGMOD International Conference on Management of Data*, 580–590, ACM, 1987. 234

[2] C. Baru, A. Gupta, B. Ludäscher, R. Marciano, Y. Papakonstantinou, and P. Velikhov. XML-based information mediation with MIX. In *Proc 1999 ACM SIGMOD International Conference on Management of Data*, 597–599, ACM, 1999. 220

[3] S. Bergamaschi, S. Castano, and M. Vincini. Semantic Integration of Semistructured and Structured Data Sources. *SIGMOD Record*, 28(1):54–59, 1999. 237

[4] J. Clark, S. DeRose. XML Path Language (XPath) Version 1.0, W3C Recommendation, Nov 1999. 225

[5] K. E. Campbell, D. E. Oliver, E. H. Shortliffe: The unified medical language system: towards a collaborative approach for solving terminology problems. *JAMIA*, Volume 8, 12–16, 1998. 223

[6] L. M. Delcambre, D. Maier, S. Bowers, M. Weaver, L. Deng, P. Gorman, J. Ash, M. Lavelle, J. Lyman: Bundles in Captivity: An Application of Superimposed Information. In Proc. of the 17th International Conference on Data Engineering (ICDE 2001), IEEE Computer Society, 111-120, 2001. 237

[7] Dublin Core Metadata Initiative, dublincore.org/. 223

[8] S. Decker, M. Erdmann, D. Fensel, R. Studer: Ontobroker: Ontology based Access to Distributed and Semi-Structured Information. In *Database Semantics*

- *Semantic Issues in Multimedia Systems, IFIP TC2/WG2.6 Eighth Working Conference on Database Semantics (DS-8)*, 351–369. Kluwer, 1999. 237

[9] D. Fensel, J. Angele, S. Decker, M. Erdmann, H.-P. Schnurr, S. Staab, R. Studer, A. Witt. On2broker: Semantic-based access to information sources at the WWW, 1999. In: Proceedings of the World Conference on the WWW and Internet (WebNet 99), 1999. 237

[10] A. Farquhar, R. Fikes, J. Rice: The Ontolingua Server: A Tool for Collaborative Ontology Construction. Technical Report KSL-96-26, Knowledge Systems Laboratory, Stanford, CA, 1996. 231

[11] J. Garfunkel: Web Annotation Technologies. look.boston.ma.us/garf/webdev/annote/software.html. 237

[12] H. Garcia-Molina, Y. Papakonstantinou, D. Quass, A. Rajaraman, Y. Sagiv, J. Ullman, V. Vassalos, and J. Widom. The TSIMMIS approach to mediation: Data models and languages. *Journal of Intelligent Information Systems*, 8(2):117–132, March 1997. 220

[13] T. R. Gruber. Toward Principles for the Design of Ontologies user for Knowledge Sharing. *International Journal on Human-Computer Studies (1993)*. 231

[14] M. Gertz, K. Sattler, F. Gorin, M. Hogarth, J. Stone: Annotating Scientific Images: A Concept-based Approach. In *14th Int. Conference on Statistical and Scientific Databases*, 59–68, IEEE Computer Society, 2002. 231, 232

[15] J. Heflin, J. Hendler: Dynamic Ontologies on the Web. In *Proc. of the 17th National Conference on Artificial Intelligence (AAAI 2000)*, 443–449, AAAI/MIT Press, 2000. 237

[16] R. M. Heck, S. M. Luebke, C. H. Obermark: A Survey of Web Annotation Systems, www.math.grin.edu/∼luebke/Research/Summer1999/survey_paper.html. 237

[17] S. Koslow, M. Huerta (eds.): *Neuroinformatics: An Overview of the Human Brain Project*. Lawrence Erlbaum Associates, NJ, 1997. 221

[18] J. Kahan, M.-R. Koivunen, E. P. Hommeaux, R. R. Swick: Annotea: An Open RDF Infrastructure for Shared Web nnotations. In *Proc. 10th International World Wide Web Conference (WWW10)*, 623–632, ACM, 2001. 237

[19] B. Ludäscher, A. Gupta, and M. Martone. Model-Based Mediation with Domain Maps. In *Proc. of the 17th Int. Conf. on Data Engineering, April 2-6, 2001, Heidelberg, Germany*, 81–90, 2001. 237

[20] E. Mena, V. Kashyap, A. Illarramendi, and A. Sheth. Domain Specific Ontologies for Semantic Information Brokering on the Global Information Infrastructure. In *International Conference on Formal Ontologies in Information Systems (FOIS'98), Trento (Italy)*, 269–283, 1998. 237

[21] Regional Primate Research Center, Neuroscience Division, University of Washington. braininfo.rprc.washington.edu/mainmenu.html. 223

[22] Neuroinformatics – The Human Brain Project. www.nimh.nih.gov/neuroinformatics/index.cfm. 221

[23] A. Ouksel and C. Naiman. Coordinating Context Building in Heterogeneous Information Systems. *Journal of Intelligent Information Systems*, 3(2):151–183, 1994. 237

[24] A. Ouksel and A. Sheth. Semantic Interoperability in Global Information Systems: A Brief Introduction to the Research Area and the Special Section. *SIGMOD Record*, 28(1):5–12, 1999. 237

[25] T. A. Phelps, R. Wilensky: Multivalent Annotations. In *Research and Advanced Technology for Digital Libraries – First European Conference*, 287–303, LNCS 1324, Springer, 1997. 237

[26] R. Stevens, P. Baker, S. Bechhofer, G. Ng, A. Jacoby, N. W. Paton, C. A. Goble, A. Brass: TAMBIS: Transparent Access to Multiple Bioinformatics Information Sources. *Bioinformatics* 16(2):184–186, 2000. 237

[27] R. Stevens, C. Goble, I. Harrocks, S. Bechhofer: Building a Bioinformatics Ontology using OIL. To appear in a special issue of IEEE Information Technology in Biomedicine on Bioinformatics, 2001. 237

[28] A. P. Sheth: *Changing Focus on Interoperability in Information Systems: From System, Syntax, Structure to Semantics*. Kluwer Academic Press, 1999. 237

[29] B. Swartout, R. Patil, K. Knight, T. Russ. Toward Distributed Use of Large-Scale Ontologies, 1996. In *Proc. 10th Knowledge Acquisition for Knowledge-Based Systems Workshop*, Alberta, Canada, 1996. 231

[30] Topic Maps. www.topicmaps.org. 237

[31] UC Davis/UC San Diego Human Brain Project Informatics of the Human and Monkey Brain, nir.cs.ucdavis.edu. 221

[32] K. Tochtermann, W.-F. Riekert, G. Wiest, J. Seggelke, B. Mohaupt-Jahr: Using Semantic, Geographical, and Temporal Relationships to Enhance Search and Retrieval In Digital Catalogs. In *Research and Advanced Technology for Digital Libraries – First European Conference, ECDL'97*, 73–86, LNCS 1324, Springer-Verlag, Berlin, 1997. 225

Exploiting and Completing Web Data Sources Capabilities

Omar Boucelma[1], Mehdi Essid[1], Zoé Lacroix[2], and Abdelkader Bétari[3]

[1] Université de Provence, LSIS
39, rue Joliot-Curie, 13453 Marseille Cedex 13, France
{omar,essid}@cmi.univ-mrs.fr
[2] Arizona State University
PO Box 876106, Tempe AZ 85287-6106, USA
zoe.lacroix@asu.edu
[3] IUT d'Aix-en-Provence - Département GTR
Avenue de Luminy, 13009 Marseille, France
betari@iut-gtr.univ-mrs.fr

Abstract. Complex applications require integration systems that access data locally or on the Web while being capable to exploit and complete the various capabilities available at integrated resources. In scientific applications such as biomedical, engineering or geographical, information systems (IS) are highly heterogeneous: they differ by their data representation and by their radically different query languages. Therefore, in addition to the common problem of data integration, provided scientific query languages shall also be integrated. In this paper we propose an approach that not only focuses on the data integration, but also addresses the integration of query capabilities available at the sources. An IS may provide a query capability inexistent at another IS, whereas two query capabilities may be similar but with slightly different semantics. We introduce the notion of *derived wrapper* that captures additional query capabilities to either compensate capabilities lacking at a source, or to adjust an existing capability in order to make it homogeneous with other similar capabilities, wrapped at other sources. The use of derived wrappers extends traditional mediation approaches.

1 Introduction

The exploitation of data available on the Internet is critical for professional activities. There has been a great effort in developing systems to support commercial and entrepreneurial activities. However, new approaches need to be developed to tackle a variety of specifications as needed to support scientific applications such as biomedical, engineering or geographical. These scientific application domains have in common that they are already poorly supported by standard database systems. Indeed, the database community had to make significant adjustments to handle properly scientific data. The main difficulties are to design appropriate data representation and to provide an adequate query language. As a result, developed systems often differ by their internal data representation and by the

S. Bressan et al. (Eds.): EEXTT and DIWeb 2002, LNCS 2590, pp. 241–258, 2003.

query functionalities made available to the users. The access to data available on the Web makes the heavily treated problem of data integration more complex by the need to exploit Web data sources capabilities. In this paper, we illustrate our approach with the problem of integrating geographic Web resources.

Geographic data have been collected for centuries and are available in a wide variety of formats and supports. Legacy data remain on paper when other have been digitized. Recent geographic data are available in flat files, databases or Geographic Information Systems (GIS), stored locally or on the Web. Most of the geographic data providers such as the French Institut National de la Statistique et des Etudes Economiques (INSEE)[1], Michelin[2] or the French Research Institute for Exploitation of the Sea (IFREMER)[3] deliver their data in ASCII files (see for example EDIGEO, the road network delivered by Michelin). There is no accepted standard for spatial or geographic data representation. Each GIS provides its own proprietary format as well as its specific query language. In addition, geographic resources are designed for a variety of different purposes. Orthogonal directions in the design of geographic resources may affect the semantics of the data they contain and impair their integration. These discrepancies make the integration of different geographic resources significantly complex.

The database community has extensively investigated architectures and tools for data integration [35] and developed data warehouses and middleware solutions to facilitate interoperability and data exchange, and heterogeneous distributed DataBase Management Systems (DBMS) or mediation techniques that facilitate data integration. However they do not address the problem to access or maintain sophisticated tools that enhance data manipulation. Indeed, in addition to standard data manipulation such as performed by SQL, geographic languages usually express spatial selections, metric or topological queries, allocations, etc. [25] and are usually implemented as ad-hoc functions with respect to a specific data representation and indices. These query capabilities may be available partially or totally at some of the data sources. Similar operators may not be semantically equivalent at two different sources. Most of the existing mediation approaches such as Tsimmis [9], Hermes [30], Information Manifold [17], or DISCO [31] focus on the data integration aspects without providing any facility to integrate available tools.

In this paper we present a mediation system that addresses the integration of GIS data and tools. It has a multi-tier client-server architecture based on the Web Feature Server (WFS) specifications [21] and uses standard wrappers to access data, extended by *derived wrappers* that capture additional query capabilities. This approach offers several advantages. Derived wrappers can either capture query capabilities available at the source or access a local query capability not available at the source in order to make it available to the user. This approach enables a rich query language in contrast to an approach that would only access

[1] http://www.insee.fr
[2] http://www.michelin.fr
[3] http://www.ifremer.fr

the query capabilities available at all sources. In addition, a derived wrapper may adjust semantic differences of similar query operators.

The paper is organized as follows. Related work is presented in Section 2. Section 3 motivates our approach with an example. Section 4 is devoted to the architecture of the GIS mediation system we propose in this paper. Finally we conclude in Section 5.

2 Related Work

2.1 Materialized vs. Non-materialized

The database community has developed two main approaches to data integration [34, 32]. In the *materialized* approach (also called data warehouse), data collected from various sources are loaded in a new system, where they are hosted for querying. The second approach is *non-materialized* (or query-driven) and data remain in the participating repositories where they are accessed and queried.

In the data warehouse approach, the main issues are data collection, warehouse organization and query processing. Expertise is hard-coded within the selection of the data sources that will provide the data, the design of the data organization and the query language. Another advantage of a data warehouse approach is the ability to control that the data loaded in the warehouse is "curated" (as curated in a museum). Indeed biomedical applications often require large amount of resources to validate the data prior to loading. The price to pay resides in the maintenance of the warehouse. Indeed, the data contained in a warehouse may become stale unless an update mechanism reactualizes the warehouse. Such mechanism is complex and costly. Nevertheless, warehouses are proven successful for scientific applications that typically handle large amount of data.

On the other hand the non-materialized approach raises various challenges including (semantic) schema integration, query re-writing and splitting, caching, and metadata management. Examples of non-materialized approaches are distributed database systems, federations, multi-database systems, and mediations. Distributed database systems consist of logically inter-related databases, distributed at multiple sites, connected by a computer network such that each database has autonomous processing capability[4] and participates in the execution of the query (split across multiple sites). A federation of databases consists in linking semi-autonomous, distributed databases. Each database has significant autonomy in the distribution while providing an interface to provide the user the capability to access integrated resources in a unified manner. In a federation, databases do not have total autonomy since they must maintain the links to the other members of the federation. A multi-database system consists in connecting fully autonomous distributed databases. Unlike the federation, in

[4] The autonomy of a DBMS expresses the degree of control of the database into the architecture, that includes what transactions are permissible, how it executes transactions, etc.

a multi-database architecture, none of the integrated resource needs to be aware of the other integrated resources, nor the overall integration component. The integration component is called a multi-database system and is in charge of (1) providing a global view of integrated resources to the user, (2) proving the user with a query language to query integrated resources, (3) execute the query by collecting needed data from each integrated resource, and (4) return the result to the user. For the user, the multi-database system provides a single view of the integrated data as is it were a single database. Finally, a mediation system (or mediator) integrates fully autonomous distributed heterogeneous data sources. Contrary to the multi-database approach, mediators do not assume that integrated sources will all be relational databases. Instead, integrated resources can be various database systems (relation, object-relational, object, XML, etc.), flat files, etc.

Since the problem of integration of Web data and capabilities involves heterogeneous systems of different kinds, we choose to use a mediator and wrappers to access and query integrated resources. In such context, when a query is issued, appropriate data sources are selected and sub-queries are generated by the mediator and sent to the corresponding information sources via their wrapper [6, 27, 5, 3]. Wrappers access data sources, retrieve and translate the results into some common integrated representation. In the context of a geo-spatial application, the mediator/wrapper approach is suitable when data needs to be up-to-date whereas the contents of integrated data sources change rapidly. A mediation approach is also more appropriate to capture and exploit applications that may be made available at the integrated sources. The approach presented in the paper is a mediation approach that aims to integrate both data and source capabilities.

2.2 Data Shipping vs. Query Shipping

One of the major issues in large-scale distributed client/server applications is the tradeoff that has to be made when designing the system's functional architecture. For Web applications, the 3-tier model is the de facto client/server model, and it is expected that most of new applications will be N-tiered. Since the domain is in constant evolution, there exist no well established techniques to analyze the performance (benchmarks) of distributed integration systems.

Once the N-tier data model is adopted, we are still left with a methodology choice between *data shipping* where data is retrieved from the server and sent to the client for processing, and *query shipping* where the query is executed on the server and results are sent to the client.

Similar issues were addressed by MOCHA [26] that aims to extend integrated resources with user-defined capabilities. They choose a *query shipping* approach by deploying the needed capabilities as Java code at the remote integrated sources. Instead, our approach is based on *data shipping*, since the queries submitted to the sources are data collecting queries, whereas the query evaluation is performed locally. In the future, the system may follow an hybrid approach combining query- with data-shipping to exploit as much as possible Web

resources in a sake of better performance. In addition, MOCHA is a "full-fledge" middleware system, while our system advocates a mediation infrastructure that complies with emerging standards and OpenGIS recommendation.

2.3 Using XML to Format Data

Since XML was designed to format data and documents for exchange, our approach naturally uses XML. Complex activities such as geographical or biomedical have devoted significant effort to homogenize their data sets with commonly accepted XML data representations. Examples include the BIOpolymer Markup Language (BIOML)[5] and the Geography Markup Language (GML) [20].

An XML format is adequate when it facilitates the tasks of a client (a mediator), that are: (1) formating the input stream for display, (2) combining it with other data streams, or (3) redirecting through a Portrayal Service. At a first glance, GML may be viewed as a means of encoding geo-spatial information, providing a great improvement to traditional reliance of GIF images for geographic maps. Indeed, a GML map encoding is a non-proprietary file that can be ported to many display systems, scaling from a Web browser to a PDA; this file can also be used for editing purposes. Given a Spatial Reference System (SRS) the GML 2.0 specification allows the definition of points, line strings and polygons. It also allows geometries that are collections of other geometries. Figure 1 illustrates a subset of a digital representation of the City of Cambridge in England as described in [20].

Using GML to handle geographic data on the Web offers several advantages [11]. The next section, will address the use of such format to facilitate the access to data. However GML can also be used as a global schema in GIS data integration applications, and thus enhances current interoperability efforts. In addition, from the user perspective, using a GML-based schema at the query interface level will hide the proprietary schemas of the remote GIS to be integrated. Many companies such as ESRI [13], Intergraph [12], or IONIC [28] are in the process to exploit further GML, by supporting WFS interfaces as defined in the next section. However none of these companies is suggesting a mediation platform.

2.4 Wrappers vs. WFS

In this section we report on our experiments is using two approaches to mediation.

In a wrapper/mediator system, a query is decomposed into sub-queries, each of them is sent to the remote data source (via its wrapper) for execution. The use of wrappers raises the following problems. First a single methodology must be capable to handle various different formats. In addition, wrappers must be maintained since data providers may change the entry points to the data sources as well as the data organization (schema). The first data integration approach

[5] See http://www.bioml.com.

```
<City xmlns:gml="http://www.opengis.net/gml">
<gml:name>Cambridge</gml:name>
<gml:boundedBy>
  <gml:Box srsName="http://www.opengis.net/gml/srs/epsg.xml#4326">

<gml:coord><gml:X>0.0</gml:X><gml:Y>0.0</gml:Y></gml:coord>

<gml:coord><gml:X>100.0</gml:X><gml:Y>100.0</gml:Y></gml:coord>
  </gml:Box>
</gml:boundedBy>
<River>
  <gml:description>The river that runs through
Cambridge.</gml:description>
  <gml:name>Cam</gml:name>
  <gml:centerLineOf>
    <gml:LineString
           srsName="http://www.opengis.net/gml/srs/epsg.xml#432">

<gml:coord><gml:X>0</gml:X><gml:Y>50</gml:Y></gml:coord>
<gml:coord><gml:X>70</gml:X><gml:Y>60</gml:Y></gml:coord>
<gml:coord><gml:X>100</gml:X><gml:Y>50</gml:Y></gml:coord>
   </gml:LineString>
  </gml:centerLineOf>
</River>
.........
<dateCreated>14/04/2002</dateCreated>
</City>
```

Fig. 1. An example of geographical data in GML format

we adopted led to the design of InterMed [2] – a mediation system for the integration of data and tools – and its application to a GIS data integration problem. A GIS mediation system prototype was built on top of LeSelect data integration engine [16]. In this architecture, a geographic SQL-like query is posed against the virtual geographic global schema, then translated into an SQL query on a virtual relational schema in order to comply with LeSelect (data integration and query) models. LeSelect allows the publication of relational data sources by registering them with a wrapper connected to a mediator. Data source wrappers are built by a *Wrapper Factory* module, using a wrapper definition file. This file is either created manually by a publisher (data sources are present when the server is launched), or automatically by a *program wrapper* (in the presence of new data sources or GIS operators). Wrapping query capabilities becomes critical in scientific applications. Building and maintaining various wrappers for scientific applications may soon become an insurmountable task.

An alternative direction uses standards to avoid the integration conflicts beforehand. The solution consists in using a common format for data representa-

tion as well as a common representation of provided data accesses. The problems mentioned above are eluded when both data and query capabilities are represented in a commonly accepted format. As part of its interoperability program, the OpenGIS consortium (OGC) is pushing towards the adoption of new standards for GIS data delivery on the Web, known in the literature as Web Map Servers (WMS) and Web Feature Servers (WFS) [21]. A WFS is a kind of GML server that allows *filter queries*, and transactions to a datastore that contains geographic *features*. In this context, a query is divided into sub-queries that are shipped to a WFS data server, allowing the retrieval of features, *query capabilities* and data. The server returns data in XML, with a structured content in GML. Adopting standards is not only a business decision but also a technology one: as an example, a WFS allows a transparent access to various GIS products, without coping with their wrapping.

Major GIS vendors, such as ESRI or Intergraph - are now offering WFS connectors to their products. Other companies – e.g., IONIC – are deploying WFS-based solutions in the context of geo-spatial applications such as the European Spatial Agency e-business application on fires' database [7, 28]. However, we believe that geo-spatial interoperability could be enhanced by combining, WFS/GML with mediation technology. Clearly, the system presented in the paper could lead to a major improvement for mediation-based systems because the burden is no more on wrapper development, and scalability is enforced since WFS-enabled Web data sources can be integrated with our system.

2.5 Towards Peer-to-Peer Sources and Resource Integration

In this section, we discuss how our approach may be generalized to peer-to-peer (P2P) data an tools integration, in particular Web Services [33].

As defined by the Peer-to-Peer Working Group,[6] P2P computing is the sharing of computer resources and services, including the exchange of information, processing cycles, cache storage, and disk storage for files, by direct exchange between systems. P2P computing approach offers various advantages: (1) it takes advantage of existing desktop computing power and networking connectivity, (2) computers that have traditionally been used solely as clients communicate directly among themselves and can act as both clients and servers, assuming whatever role is most efficient for the network, and (3) it can reduce the need for IT organizations to grow parts of its infrastructure in order to support certain services, such as backup storage [23].

This approach to better exploitation of available resources in the context of enterprise is similar to a concurrent effort: the GRID,[7] to design better solutions

[6] Information about the Peer-to-Peer Working Group is available at http://www.peer-to-peerwg.org/.

[7] The Global Grid Forum (GGF) is a community-initiated forum of individual researchers and practitioners from over 200 organizations in over 30 countries working on distributed computing. Information about GGF is available at http://www.ggf.org/.

to provide to the scientific community, industry, government and the public at large dependable, consistent, pervasive and inexpensive access to high-end computational capabilities [10]. These two efforts recently merged. Popular applications of P2P computing include file sharing products such as Morpheus [18], Kazaa Media Desktop [14] and Napster [22] or more advanced academic prototypes such as OceanStore [19].

P2P computing alleviates some of the drawbacks encountered in centralized coordination, mainly scalability and availability of services. In contrast, the presented WFS-based integration approach is promoting a classical centralized architecture. Indeed, the mediator plays a central role in performing data integration/mediation, query dispatching and so forth. However, in adopting a cascading WFS architecture [21], each participating node – e.g., a GIS or the mediator – may be WFS enabled. In order to provide a full P2P architecture, each peer should also play the following roles [24]: user of services, provides services, forward requests, and cache information. Web services [33] provide UDDI for service discovery, WSDL for service specification, and SOAP as a messaging protocol. We believe that Web services are a promising technology that may leverage the design of P2P architectures in the future. Indeed, in turning WFS to a compliant web service, we obtain a general P2P architecture for data sources and resources integration. Similar integration was proposed with Active XML (AXML) [1] that integrates P2P data and Web services.

3 Motivating Example

Spatial integration is predominantly performed manually, using important human resources [8]. Most current approaches to geographic data integration are data warehouses. This paper does not address the specific issue of schema mapping for data integration. It rather focuses on issues related to the expressive power of query languages.

3.1 Source Schemas

In the sequel, we use an example drawn from [8] which is inspired from the cartographic and topographic databases maintained by the French National Geographic Institute. This example describes two road networks S1 and S2 describing the same geographic area. Source S1 is equivalent to a 1:250'000 scale, while S2 is more detailed and is equivalent to 1:10'000 scale. Issues raised by the integration scenario are discussed in [8], while more detailed issues on GIS integration may be found in [15].

Figure 2 shows a UML diagram for data sources S1 and S2, together with the integrated schema (IS), as described in [8].
Data source S1 contains the following object types:

– *RoadSection* describes a piece of road, homogeneous in value with respect to the set of attributes and relationships. For instance, changing the number of lanes of a road, results in the creation of a new road section.

– *Node* describes points in the road network where traffic conditions change, hence delimiting road sections.

Data source S2 contains the following object types:

– A *WaySection* is composed by one or several contiguous lane(s) going in the same direction. If the road has several lanes in the same direction split by a separator, then each group of lanes is represented as a separate way section. For instance, a bus lane may be physically separated from two car lanes.
– A *Separator* describes separators between two way sections of the same road.
– An *Extremity* is a point where traffic conditions change. It is either a crossroad, a traffic circle, a toll, a dead end, a value-change node or an end of tunnel. Because the subtypes (crossroad, toll, end of tunnel) have different geometries, the type of an extremity is denoted by geo (any geometry).

3.2 Schema Integration

Consider the XQuery expression (Q) that *"extracts all WaySections included in the RoadSection "R1""* as defined against the integrated schema in Figure 3. Query (Q) involves two Inter-schema Correspondence Assertions (ICA): C1 and C2, that express inter-schema relationships. More details on these assertions are given in [8]. Clause (C1) specifies that every RoadSection instance corresponds

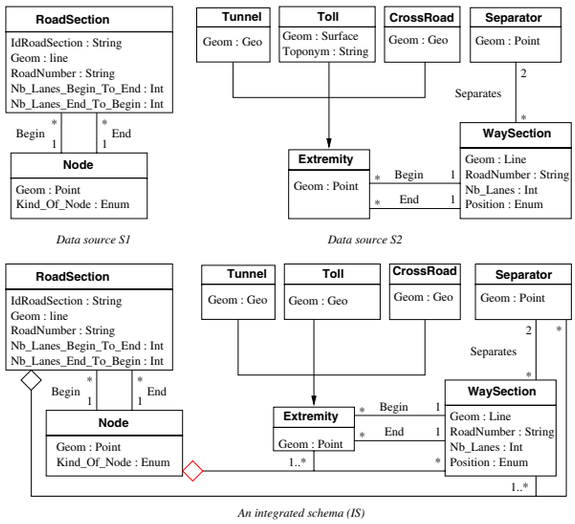


Fig. 2. The database schemas

XQuery (Q):
FOR $W IN document("WaySections")
FOR $R IN document("RoadSections")
WHERE $W/InRoadSection = $R/IdRoadSection
AND $R/IdRoadSection = "R1"
RETURN
<WaySections>$W</WaySections>

Clause (C1):
RoadSection ⊆ SET([1:N]WaySection, [0:N]Separator)

Clause (C2):
WaySection.RoadNumber = RoadSection.RoadNumber
AND WaySection.geo INSIDE
BUFFER(RoadSection, resolution)

Fig. 3. Query (Q) involving ICA rules C1 and C2

to a multi-sorted set of (WaySection,Separator) instances. Clause (C2) specifies that, for each RoadSection instance, only WaySection instances that satisfy the two conditions below should be considered: (1) match the RoadNumber attribute, and (2) whose geometry lies within a given BUFFER surface enclosing the road section geometry.

The INSIDE operator is the topological relationship which checks whether the point locating the crossroad from S2 is within the area computed by the BUFFER function, which transforms the point representing the S1 node into a surface geometry according to the resolution of data source S2.

Note that query (Q) uses (an additional) attribute IdRoadSection that results from a UML to semi-structured transformation of the integrated schema. This attribute expresses the fact that a road section (in S1) corresponds to a set of way sections (in S2).

During the integration process, information is provided either by the data sources, or in computing some operators such as INSIDE: this is a kind of "impedance mismatch" in the sense that there is no data source INSIDE(x,y) to pull data from. Should such a data source be available, the data integration problem would be treated in a uniform way with respect to the mediation solution we discuss in this paper. Our solution relies on a mediation system where resources (operators) are treated as virtual data sources by means of *derived wrappers*.

4 Geographic Mediation System

In this section we discuss the architecture of our geographic mediation system.

4.1 Mediation Architecture

Figure 4 describes the functional architecture of the geographic mediation system which is mainly composed of three layers: a GIS mediator, Web Feature Servers (WFS) [21] and data sources. The GIS mediator is composed of a query processor which consists of three components: an *Analyzer*, an *Optimizer*, and an *Execution* module. The GIS mediator is in charge of analyzing the (geographic) query, including various transformations that involve ICA rules, performing some optimizations, and splitting the query into sub-queries, passing them to the right WFS for execution.

The WFS layer is in charge of manipulating the data sources: it receives requests from the mediator, executes them, and returns the results.

A WFS request consists of a description of a query or data transformation operations that are to be applied to one or more features. The request is issued by the client and is posted to a Web server using HTTP. The WFS is invoked to service the request. A request combines operations defined in the WFS specification. Among those operations are:

- `GetCapabilities`: a WFS must be able to describe its capabilities. Specifically, it must indicate which feature types it can service and what operations are supported on each feature type.
- `DescribeFeatureType`: a WFS must be able, upon request, to describe the structure of any feature it can service.
- `GetFeature`: a WFS must be able to service a request, and to retrieve feature instances.

`GetFeature` requests (posted by the *Analyzer* after transformations) are illustrated in Figure 6. Nowadays, the popularity of XML as a data exchange format, GML designed to represent geographic data, and XQuery [4] makes them good candidates to express respectively schemas, data and queries. The WFS reads and executes the request and returns an answer to the user as a GML or XML document depending on the request.

In Section 1, we have motivated the need for extending standard data integration approaches in order to cope with query capabilities. These query capabilities may need semantic adjustment (when two similar capabilities are not semantically equivalent) or added (when they are not available at a given source). In addition, ICAs may involve operators (such as INSIDE) that are not defined at any data source. Each of these capabilities or integration operators has an implementation locally stored in the *Operators* repository. Its execution is performed by a *Program Wrapper* that creates a virtual additional data source and a WFS which allows the mediator to access to it.

In the sequel, we illustrate the analysis, optimization and execution of query (Q) defined in Figure 2.

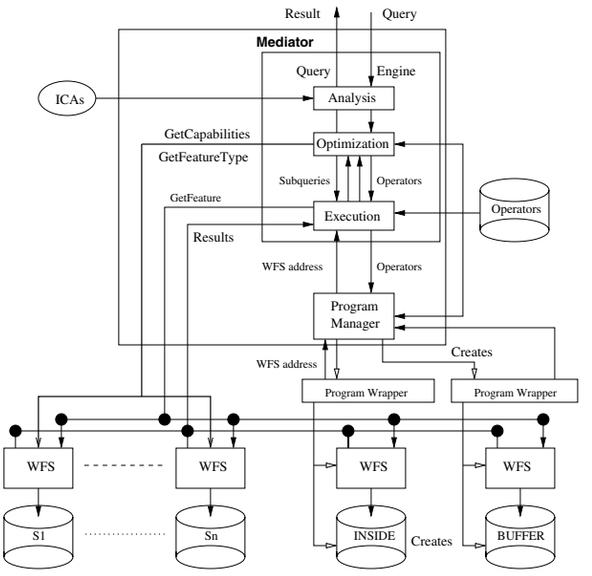


Fig. 4. GIS Mediation Architecture

```
FOR $W IN document("WaySections")
FOR $R IN document("RoadSections")
WHERE $W/RoadNumber = $R/RoadNumber
AND INSIDE($W/Geom , BUFFER($R/Geom , resolution))
AND $R/IdRoadSection = "R1"
RETURN
<WaySections>$W</WaySections>
```

Fig. 5. Query (Q') after transformations

4.2 Query Analysis

The *Analyzer* module uses the ICA rules defined in Section 3 to solve aggregation conflicts. The resolution consists in rewriting the query to eliminate the conflicts and to express it against the local source schemas. In our example, we consider two ICA clauses (C1) and (C2). Using (C1), the analyzer discovers the presence of a conflict (the fact that road sections are composed of way sections) in query (Q). This conflict refers to the use of the IdRoadSection attribute which is not available at any local source schema, therefore this attribute needs to be replaced with existing local source schema attributes. This task is performed by using clause (C2) that expresses how to solve the conflict(s) introduced by (C1). The expression containing attribute IdRoadSection in query (Q) is then replaced using (C2). This rewriting process results in an equivalent query (Q') as illustrated in Figure 5.

4.3 Query Optimization

The *Optimizer* component exploits the information about source feature types and WFS capabilities to select an efficient execution plan by dividing the query in several sub-queries. The needed information is provided by WFS in the form of XML and XML schema documents, each time they receive *GetCapabilities* or *GetFeatureType* requests. Using an XQuery processor, the *Optimizer module* establishes the execution plan of the query. It splits query (Q') into two kinds of sub-queries: (1) pure XQuery queries that do not contain operators, and (2) operator queries [8] that cannot be *supported* [9] by integrated data sources.

All sub-queries are treated similarly by the WFS layer. A sub-query requiring an operator that is not wrapped at any of the integrated sources is processed by the execution module to create a *derived wrapper* implemented as a WFS as explained in Section 4.4. Since we propose a WFS based architecture, each of the sub-queries is then translated into the query language supported by the WFS, i.e., XML+KVP (keyword-value pair) [21]. An example of a KVP is: `"REQUEST=GETCAPABILITIES"`.

In our example, query (Q') is divided into two sub-queries: Q1 and Q2 to be directly shipped to the WFS that capture data sources S1 and S2, and into parameterized queries: Q3 and Q4 that represent the operators INSIDE and BUFFER locally available. Queries Q3 and Q4 represent XML+KVP document queries that result from a translation of operator queries. XML+KVP queries Q1, Q2, Q3 and Q4 are illustrated in Figure 6.

The mediation architecture provides the ability to improve the optimization phase by using additional (metadata) information about sources and operators' capabilities. Identifying resource characteristics that can be exploited by the users and the query planning component is critical for the efficiency of the mediation system. We aim to identify metadata along several dimensions, including: (i) the coverage of the information sources, (ii) the capabilities, and (iii) the data delivery patterns of the resources. In the future, useful such metadata could be included in the features supported by WFS.

The step-by-step processing of a query is illustrated in Figure 7. Once the whole rewriting process is done, queries Q1, Q2, Q3, and Q4 are executed as follows.

Sub-queries Q1 and Q2 invoke respectively S1 and S2 to retrieve road sections and way sections. Q1 and Q2 are directly sent to the appropriate WFS by the execution module. When the results of Q1 and Q2 are returned, they are saved respectively in (temporary) QWaySections and QRoadSections, GML structures (respectively noted C and T in Figure 7), allowing the system to execute operation BUFFER(QRoadSections,resolution). The *Optimizer module* creates a parameterized XML+KVP document query, then pass the operation to the *Execution module* which executes it and sends to the *Optimizer* needed

[8] Operator queries are queries containing only specific operators such as INSIDE, BUFFER, etc.

[9] A data source is viewed as a DBMS and its instance.

```
< GetFeature >                                          (Q1)
   < QueryTypeName = "WaySection"/ >
< /GetFeature >

< GetFeature >                                          (Q2)
   < QueryTypeName = "RoadSection"/ >
   < Filter >
         < PropertyIsEqualTo >
            < PropertyName >
               RoadSection.IdRoadSection
            < /PropertyName >
            < Literal > R1 < /Literal >
         < /PropertyIsEqualTo >
   < /Filter >
< /GetFeature >

< GetFeature >                                          (Q3)
   < QueryTypeName = "BUFFER"/ >
< /GetFeature >

< GetFeature >                                          (Q4)
   < QueryTypeName = "INSIDE"/ >
< /GetFeature >
```

Fig. 6. Query decomposition

information to extract the results. The way the *execution module* executes operator queries and information returned by the *Optimizer* is detailed in the next paragraph. The optimizer finally builds a XML+KVP request Q3 and sends it to a *derived wrapper* implemented by a WFS that will service the request in returning results (noted T' in Figure 7) to the optimizer module. The INSIDE operator and query Q4 are treated in the same way. Finally, the answer to the original query is built and sent to the user.

4.4 Query Execution

The *Optimizer* processes the user's query and forwards the generated sub-queries directly to the *Execution* module as described in Section 4.3. The *Execution* module executes the sub-queries then it integrates the results returned by the different WFS.

The *Execution* module processes sub-queries as follows. Sub-queries that correspond to existing sources are directly sent to the appropriate WFS. Sub-queries requiring an operator that is not available at any of the integrated sources is processed differently. As each spatial operator is implemented by a *Program Wrapper*, the *Execution* module triggers a job request P1 to execute the operation. Program P1 is processed by the *Program Manager* component which, in turn,

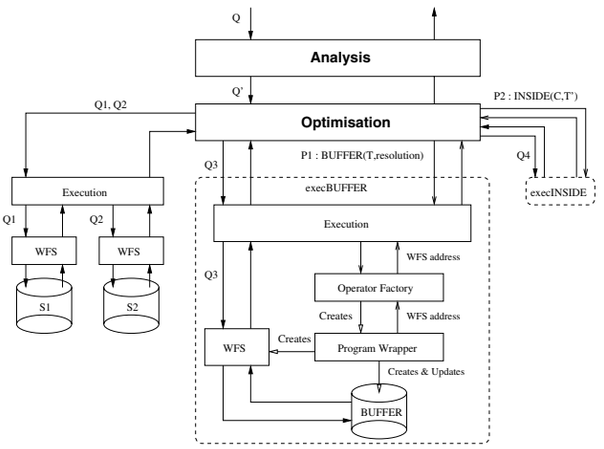


Fig. 7. Query Execution

builds the associated *Program Wrapper*. A virtual data source is created and populated by results returned by program P1. A *derived wrapper*, implemented as a WFS, is also created in order to access this new data source. Finally, the WFS address is sent to the *Optimizer* module to extract results. It is worth noting that operator sources are accessed through WFS as regular integrated sources.

The architecture described above has several advantages. It first treats equally data generated by applications or contained in data sources. In scientific applications such as geography, it is critical since data is often generated by complex tools as opposed to be hosted in a data source. It is also easy to use the mediation upon sources that are not databases or GIS but can be flat files or Web-hosted sources. Again, in scientific domains, most of the data is actually not stored in database systems. A second advantage of the approach is the use of *derived wrapper* to host operators that enable the extension of the global mediated system with functionalities that may be missing at some or all integrated sources. With this architecture all data generated by local or remote applications or stored in databases is accessed the same way through WFS. A third advantage is to have a distributed system. Indeed each sub-query submitted by the mediator is considered as a WFS client. Since WFS uses HTTP as a communication protocol, data sources and the mediator may not reside on the same server. Finally, the mediation approach uses commonly accepted format and language, and facilitates maintenance.

4.5 Implementation Details

We are using the GeoServer [29] free implementation of OpenGIS Consortium's WFS implementation specification. As mentioned in Section 4.4, the system uses two kinds of WFS servers: static and dynamic. Static servers handle for available data sources, and dynamic servers handles host (derived) data that are dynamically generated. Each server is implemented by means of a GeoServer instance. To better understand our implementation, let us consider the processing of query Q3 described above. A program wrapper creates a new (derived) data source and its corresponding WFS/GeoServer instance (the server). This is done in three steps:

1. Generating a XML configuration file that contains the parameters that are mandatory for a GeoServer instance: server name, its address, etc.
2. Binding (connecting) the new data source to the server. This is done by creating a dedicated directory schema information and metadata useful for each table in this data source[10]. For example, the following information will be associated to a table `<myTable>`: the `schema.xml` file is the GML description of the source schema, the `info.xml` file provides `<myTable>`'s address, a Spatial Reference System (SRS), etc.
3. Instantiating the server. The WFS/GeoServer is now ready to accept queries.

5 Conclusion

The Web offers numerous resources that not only provide data but also sophisticated applications to access, manipulate, analyze and visualize the data they contain. Integrating Web data only, while not exploiting these useful resources, would be of little interest. The system presented in the paper aims to integrate with a non-materialized mediation, Web data and applications. The approach exploits Web standards and uses traditional wrappers and *derived wrappers*. Wrappers exploit available resources when derived wrapper replace missing capabilities, as long as there are available locally or elsewhere on the Web and are used to reconcile two similar query capabilities but with a slightly different semantics. The use of Web standards and derived wrappers extends traditional mediation approaches and provides a framework to integrate Web resources for complex tasks, such as scientific applications.

The presented system was implemented and demonstrated in the context of geo-spatial applications.

Acknowledgment

The work presented in the paper was partially supported by a grant of the French Ministère de l'éducation nationale, de la recherche et de la technologie for the programme Réseau National de recherche et d'innovation en Technologies Logicielles (RNTL).

[10] We are using relational tables to host new data sources.

References

[1] S. Abiteboul, O. Bendjelloun, I. Manolescu, T .Milo, and R. Weber. Active XML: Peer-to-Peer Data and Web Services Integration. In *VLDB2002*, Hong Kong, China, 2002. Demo. 248

[2] O. Boucelma and Z. Lacroix. InterMed : une interface de médiation pour les systèmes d'information. *ISI - Hermes*, 6:33–60, december 2001. 246

[3] L. Bright, J-R. Gruser, L. Raschid, and M. E. Vidal. A Wrapper Generation Toolkit to Specify and Construct wrappers for Web accessible Data Sources (WebSources). *Journal of Computer Systems Science & Engineering. Special Issue: Semantics on the World Wide Web*, 14(2), March 1999. 244

[4] D. Chamberlin, D. Florescu, J. Robie, J.Siméon, and M. Stefanescu. *XQuery: A Query Language for XML*. W3C, 2000. available at http://www.w3.org/TR /xmlquery. 251

[5] S. Cluet, C. Delobel, J. Simeon, and K. Smaga. Your mediators need data conversion. In *Proccedings of the ACM SIGMOD Conference*, pages 177–188, 1998. 244

[6] W. F. Cody, L. M. Haas, W. Niblack, M. Arya, M. J. Carey, R. Fagin, D. Lee, D. Petkovic, P. M. Schwarz, J. Thomas, M. Tork Roth, J. H. Williams, and E. L. Wimmers. Querying Multimedia Data from Multiple Repositories by Content: The Garlic Project. In *IFIP 2.6 Third Working Conference on Visual Database Systems (VDB-3)*, Lausanne, Switzerland, March 1995. http://www.almaden.ibm.com/cs/garlic. 244

[7] V. Dessard. GML & Web Feature Server. The Baseline for Online Geoservices. *GEOINFORMATICS*, pages 38–40, Mars 2002. 247

[8] T. Devogele, C. Parent, and S. Spaccapietra. On Spatial Database Integration. *International Journal of Geographical Information Science*, 12(4):335–352, 1998. 248, 249

[9] H. Garcia-Molina, Y. Papakonstantinou, D. Quass, A. Rajaman, Y. Sagir, J. Ullman, V. Vassalos, and J. Widom. The TSIMMIS approach to mediation: Data models and Languages. *Journal of Intelligent Information Systems*, 1997. (See also http://www-db.stanford.edu/tsimmis/). 242

[10] Global Grid Forum (GGF). Global grid forum overview. http://www.ggf.org/. 248

[11] Galdos Systems Inc. http://www.galdosinc.com/. 245

[12] Intergraph. http://www.intergraph.com/. 245

[13] ESRI Interoperability and Standards. http://www.esri.com/software/opengis/-interopdownload.html. 245

[14] Kazaa Media Desktop. http://www.kazaa.com. 248

[15] R. Laurini. Spatial Multidatabase Topological Continuity and Indexing: a Step towards Seamless GIS Data Interoperability. *International Journal of Geographical Information Sciences*, 12(4):373–402, june 1998. 248

[16] LeSelect. http://www-caravel.inria.fr/LeSelect/. 246

[17] A. Levy, A. Rajaraman, and J. Ordille. The World Wide Web as a Collection of Views: Query Processing in the Information Manifold. In *VIEWS96 – Workshop on Materialized Views (in cooperation with SIGMOD 1996)*, 1996. 242

[18] Morpheus 2.0 file sharing product. http://www.morpheus-os.com. 248

[19] OceanStore Project. http://oceanstore.cs.berkeley.edu. 248

[20] OpenGIS. Geography Markup Language (GML) 2.0. see http://www.opengis.net/gml/01-029/GML2.html. 245

[21] OpenGIS. OGC Request 13: OpenGIS Web Feature Server Implementation Specification. see http://www.opengis.org/info/techno/rfp13info.htm. 242, 247, 248, 251, 253
[22] OpenNap: Open Source Napster Server. http://opennap.sourceforge.net/. 248
[23] Peer-to-Peer Working Group. What is peer-to-peer? http://www.peer-to-peerwg.org/whatis/index.html. 247
[24] PEPITO - PEer-to-Peer: Implementation and TheOry. http://www.sics.se/pepito. 248
[25] Ph. Rigaux, M. Scholl, and A. Voisard. *Spatial Databases - with applications to GIS*. Morgan Kaufmann, 2001. 242
[26] Manuel Rodriguez-Martinez and Nick Roussopoulos. MOCHA: A Self-Extensible Database Middleware System for Distributed Data Sources. *SIGMOD Record*, 29(2):213–224, 2000. 244
[27] M. T. Roth and P. Schwarz. Don't Scrap It, Wrap It! A wrapper Architecture for Legacy Data Sources. *Proceedings of the International Conference on Very Large DataBases*, pages 266–275, 1997. 244
[28] IONIC Software s.a. http://www.ionicenterprise.com/. 245, 247
[29] SourceForge.net. The GeoServer Project. see http://geoserver.sourceforge.net. 256
[30] V. S. Subrahmanian, S. Adali, A. Brink, R. Emery, J. J. Lu, A. Rajput, T. J. Rogers, R. Ross, and C. Ward. HERMES: A HEterogeneous Reasoning and MEdiator System. TR available at http://www.cs.umd.edu/projects/hermes/. 242
[31] A. Tomasic, L. Raschid, and P. Valduriez. Scaling access to distributed heterogeneous data sources with disco. *IEEE Transactions On Knowledge and Data Engineering*, 10(5):808–823, 1998. 242
[32] A. Voisard and M. Jurgens. GEOSPATIAL INFORMATION EXTRACTION: QUERYING OR QUARRYING? In M. Goodchild and M. Egenhofer and R. Fegeas and C. Kottman, editor, *INTEROPERATING GEOGRAPHIC INFORMATION SYSTEM*, pages 165–179. Kluwer Academic Publishers, Boston, 1999. 243
[33] Web Services Activity. http://www.w3.org/2002/ws. 247, 248
[34] J. Widom. Research problems in data warehousing. In *Proceedings of CIKM'95*, pages 25–30, 1995. 243
[35] G. Wiederhold. Mediators in the Architecture of Future Information Systems. *IEEE Computer*, 25(3):38–49, March 1992. 242

Author Index