XML Schema Mappings for Heterogeneous Database Access ⋆

Samuel Robert Collins, Shamkant Navathe, Leo Mark

College of Computing
Georgia Institute of Technology
Atlanta, Georgia 30332, USA

Abstract

The unprecedented increase in the availability of information, due to the success of the World Wide Web, has generated an urgent need for new and robust methods that simplify the querying and integration of data. In this research we investigate a practical framework for data access to heterogeneous data sources. The framework utilizes the eXtensible Markup Language (XML) Schema as the canonical data model for the querying and integration of data from heterogeneous data sources. We present algorithms for mapping relational and network schemas into XML schemas using the relational mapping algorithm. We also present **libSyD** (library System of Databases), a prototype of a system for heterogeneous database access.

Key words: Multidatabases, Heterogeneous Databases access, XML Schema, Schema Integration

1 Introduction

XML [1] is quickly emerging as the standard for data exchange on the World Wide Web. Its potential has sparked a flurry of activity in the business and research communities. Businesses are racing to develop applications that utilize XML for business-to-business (B2B) transactions. Researchers are developing new and innovative methodologies to employ and enhance the features of XML. In this research we propose a practical framework for heterogeneous database access. We use the XML Schema [2–4] as the canonical data model

⋆ This research is part of the **XMLApe** project.

Email addresses: `scollins@cc.gatech.edu` (Samuel Robert Collins), `sham@cc.gatech.edu` (Shamkant Navathe), `leomark@cc.gatech.edu` (Leo Mark).

for a schema integration technique to facilitate the querying and integration of data from various data sources. A crucial point of schema integration is to overcome the semantic heterogeneity of the schemata to be integrated [5]. If we have data source schemas in a canonical data model, we can provide users a single uniform interface to facilitate data integration and querying without changing the underlying data sources.

This proposed research utilizes XML schemas as the unifying data model for data integration. It is greatly dependent on the mapping of data source schemas into XML schemas. In section 3, two algorithms are presented for mapping relational model and network model schemas into XML schemas. In section 4, we present libSyD, a system which allows easy access to data that resides in a library system of databases. libSyD was implemented using the relational mapping allgorithm presented in this paper.

2 Querying Heterogeneous Data

The motivation for this research stems from our ongoing efforts to develop more robust methods for querying and integrating data from heterogeneous data sources. The unprecedented increase in the availability of information due to the success of the World Wide Web has generated an urgent need for new and robust methods that simplify the querying and integration of data.

A lot of research in the past and present has focused on developing methodologies for querying heterogeneous data sources. The intent is to integrate data from existing databases in a distributed environment while minimizing the impact of operations on the databases [6]. One approach is to use a unified global integration schema, such as the relational schema, to facilitate efficient global processing. Pegasus [7], DAVID [8], MERMAID [9], CI [10], and [11] are all examples of systems, which promote this type of methodology. These systems utilize schema integration techniques, similar to those presented in [12–14,4], to efficiently query and integrate data. But, their global schemas become hard to manage as the number and types of data sources increase.

Another approach for querying data sources involves a system based on mediators and wrapper. Wrapper-mediator systems are sophisticated applications that abstract the data source from the users. In addition they translate queries into the terms of the data sources and integrate the results. SIMS [15] and [16] are examples of this type of system. The wrapper-mediator approach is remarkably scalable, and allows the integration of an increasing number of data sources.

This research is similar to the work done with mediators and wrappers as well

as global schemas. Our approach utilizes XML Schemas as the unifying data model. It is similar to work presented in [17].

3 Mapping to XML Schema

A crucial objective of schema integration is to overcome the semantic heterogeneity of the schemas to be integrated. In order to create a unified interface to the heterogeneous sources we must first develop a set of schema and data mappings from the existing databases to XML. In the following sections we concentrate on the translation of schemas from the network, and relational databases into XML schemas.

3.1 Relational to XML Mapping Algorithm

In this section we assume that the database is stored in a relational database management system (DBMS). Today, it is almost a certainty that any large commercial application would be implemented using some relational DBMS. The dominant relational DBMSs include ORACLE, DB2, INFORMIX and SYBASE. Out of these, DB2 is the product that is the oldest and has been around since 1982. Every relational DBMS stores a database in the form of relations (informally tables) which are defined over a set of domains (columns). Each relation is populated with a set of tuples (records, or rows of table). In the following algorithm, we use the above terminology for referring to the constructs of the relational model.

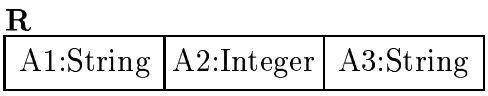
STEP 1: The first step of the algorithm is to create a schema tag with the correct XML Namespace [18] information.

```
<?xml version="1.0" encoding="UTF-8"?>
<schema
    targetNamespace="http://www.example-uri.com/Sample-DB"
    xmlns="http://www.w3.org/2001/XMLSchema"
    xmlns:dbns="http://www.example-uri.com/Sample-DB"
    elementFormDefault="unqualified"
    attributeFormDefault="unqualified">

</schema>
```

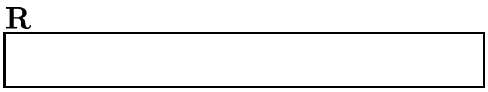
STEP 2: The next step of the algorithm is to create a complex type for each relation to hold the individual records for that relation. Therefore, for each relation R with attributes $A1 \ldots An$, create a complex type R–$RecType$, and include $A1 \ldots An$ as elements with correct simple types. In

addition, if $A1 \ldots An$ is allowed to be $NULL$, add the attribute *nillable* to the corresponding element and set its value equal to true.



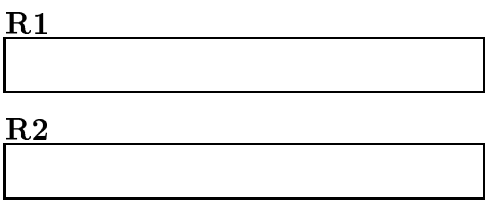

```
<complexType name="R-RecType">
    <sequence>
        <element name="A1" type="string" />
        <element name="A2" type="integer" />
        <element name="A3" type="string" />
    </sequence>
</complexType>
```

STEP 3: For each relation R, create a complex type R–$RelType$, and include an element R of type R–$RecType$. Set the $minOccurs$ attribute to 0 and $maxOccurs$ attribute to *unbounded* for each element.




```
<complexType name="R-RelType">
    <sequence>
        <element name="R" type="dbns:R-RecType"
            minOccurs="0" maxOccurs="unbounded" />
    </sequence>
</complexType>
```

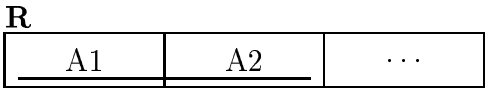
STEP 4: In this step the database element is created. For the database DB with relations $R1 \ldots Rm$, add a DB element to schema and insert an anonymous complex type. Then, include $R1$–$Rel \ldots Rm$–Rel as elements of corresponding complex types $Ri \ldots RelType$.




```
<element name="DB">
    <complexType>
        <sequence>
```

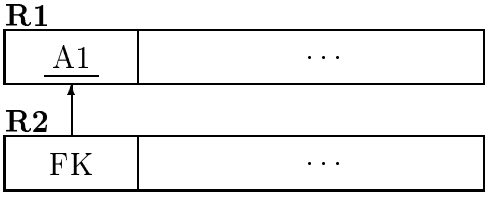
```
            <element name="R1-Rel" type="dbns:R1-RelType" />
            <element name="R2-Rel" type="dbns:R2-RelType" />
        </sequence>
    </complexType>
</element>
```

STEP 5: For each relation R, if a primary key exists with the attributes $A1 \ldots Aj$, insert a key tag $R\text{–}Key$ to the database element. Add a selector tag with the $xpath$ value set to $dbns : R\text{–}Rel/dbns : R$ to the key tag. In addition, insert field tags for the attributes $A1 \ldots Aj$.




```
<key name="R-Key">
    <selector xpath="dbns:R-Rel/dbns:R" />
    <field xpath="@A1" />
    <field xpath="@A2" />
</key>
```

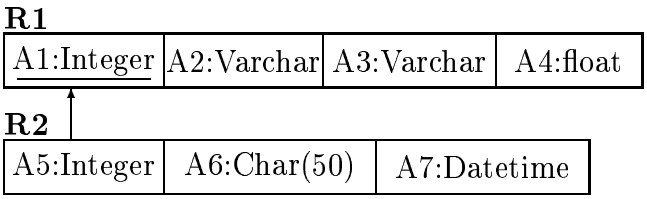
STEP 6: For each relations R with foreign keys $FK1 \ldots FKn$, insert a $keyref$ tag $R\text{–}FKi$ for each foreign key. Insert a selector tag with the $xpath$ value set to $R\text{–}Rel/R$ and field tag(s) with the $xpath$ value set to $@FK$.




```
<keyref name="R2-FK" refer="R1-Key">
    <selector xpath="dbns:R2-Rel/dbns:R2" />
    <field xpath="@FK" />
</keyref>
```

3.1.1 Relational Mapping Example

In this section we illustrate how the algorithm can be applied to a sample relational database. The sample relational schema is shown below in figure 15. The resulting XML schema, Sample-DB.xsd is also shown.



Sample-DB.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
    targetNamespace="http://www.example-uri.com/Sample-DB"
    xmlns:dbns="http://www.example-uri.com/Sample-DB"
    elementFormDefault="unqualified"
    attributeFormDefault="unqualified">
    <element name="Sample-DB">
        <complexType>
            <sequence>
                <element name="R1-Rel" type="dbns:R1-RelType" />
                <element name="R2-Rel" type="dbns:R2-RelType" />
            </sequence>
        </complexType>
        <key name="R1-Key">
            <selector xpath="dbns:R1-Rel/dbns:R1" />
            <field xpath="@A1" />
        </key>
        <key name="R2-Key">
            <selector xpath="dbns:R2-Rel/dbns:R2" />
            <field xpath="@A5" />
            <field xpath="@A6" />
        </key>
        <keyref name="R2-A5" refer="dbns:R1-Key">
            <selector xpath="dbns:R2-Rel/dbns:R2" />
            <field xpath="@A5" />
        </keyref>
    </element>
    <complexType name="R1-RelType">
        <sequence>
            <element name="R1" type="dbns:R1-RecType"
                minOccurs="0" maxOccurs="unbounded" />
        </sequence>
    </complexType>
    <complexType name="R2-RelType">
        <sequence>
            <element name="R2" type="dbns:R2-RecType"
                minOccurs="0" maxOccurs="unbounded" />
        </sequence>
```

```
    </complexType>
    <complexType name="R1-RecType">
        <sequence>
            <element name="A1" type="int" />
            <element name="A2" type="string" />
            <element name="A3" type="string" />
            <element name="A4" type="float" nillable="true" />
        </sequence>
    </complexType>
    <complexType name="R2-RecType">
        <sequence>
            <element name="A5" type="int" />
            <element name="A6" type="string" />
            <element name="A7" type="date" />
        </sequence>
    </complexType>
</schema>
```

3.2 Network to XML Schema Algorithm

This section deals with mapping schemas of databases stored in the network data model [19]. This model was endorsed by the Database Task Group [20] of ANSI back in 1971 and have resulted in a number of network DBMS implementations. Popular products include IDMS (Computer Associates), IDS II (Bull, France), IMAGE (Hewlett Packard), VAX-DBMS (DIGITAL) and DMS 1100 (Unisys). The network data model uses two basic constructs: Record Types and Set Types. A record contains data items which may be of type repeating group (multivalued item) or a vector (a list). Relationships are of type one-to-many with the one side being the owner record type and the many side being the member record type. Further information on this model may be found in [19].

STEP 1: The first step of the algorithm is to insert schema tags with the correct XML Namespace information.

```
<?xml version="1.0" encoding="UTF-8"?>
<schema
    targetNamespace="http://www.example-uri.com/Sample-DB"
    xmlns="http://www.w3.org/2001/XMLSchema"
    xmlns:dbns="http://www.example-uri.com/Sample-DB"
    elementFormDefault="unqualified"
    attributeFormDefault="unqualified">

</schema>
```

STEP 2a: The next step of the algorithm is to create a complex type to for the individual records of the various record types. Therefore, for each record type RT with data items $D1 \ldots Dn$, create a complex type $RT–RecType$, and include $D1Dn$ as elements. If Dx is a vector observe *Step 2b*. If $Dx4$ is a repeating group observe *Step 2c*.

RT	
$D1$	$D2$

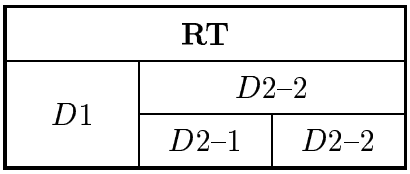
```
<complexType name="RT-RecType">
    <sequence>
        <element name="D1" type="string" />
        <element name="D2" type="integer" />
    </sequence>
</complexType>
```

STEP 2b: For the data item D, if D is a vector, add an anonymous complex type to the element $D4$. Insert element $D - Item$ and set the *minOccurs* attribute to 0 and the *maxOccurs* attribute to *unbounded*.

RT
D:Vector

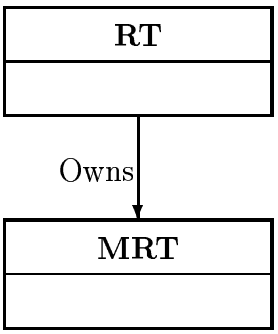
```
<complexType name="RT-RecType">
    <sequence>
        <element name="D">
            <complexType>
                <sequence>
                    <element name="D-Item" type="string"
                        minOccurs=0 maxOccurs=unbounded />
                </sequence>
            </ complexType>
        </element>
    </sequence>
</complexType>
```

STEP 2c: For each data item D, if D is a repeating group with data items $D1 \ldots Dn$, add an anonymous complex type to the element D. If Dx is a vector or repeating group, recursively call *Step 2b* and *Step 2c* for each data item that is a vector or repeating group.




```
<complexType name="RT-RecType">
    <sequence>
        <element name="D1" type="string" />
        <element name="D2"
                minOccurs=0 maxOccurs=unbounded>
            <complexType>
                <sequence>
                    <element name="D2-1" type="string" />
                    <element name="D2-2" type="Integer" />
                </sequence>
            </complexType>
        </element>
    <sequence>
</complexType>
```

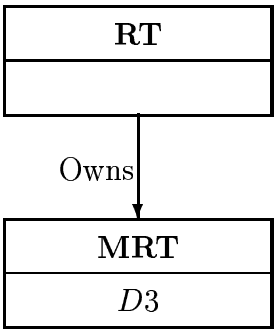
STEP 3: The attribute of type ID is used to identify all instances of owner records. Hence, for each owner record type RT insert an attribute tag RT–$ID4$ of $type$ = "ID" into the RT–$RecType$ complex type.




```
<complexType name="RT-RecType">
    <sequence>
        <element name="D1" type="string" />
        <element name="D2" type="integer" />
        <attribute name="RT-ID" type="ID" />
    </sequence>
</complexType>
```

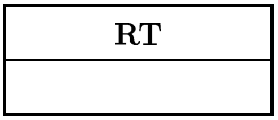
STEP 4: Member records identify their owner records with attributes of type $IDREF$. These $IDREF$ attributes reference an owner record ID. Hence,

for each member record type with set types $S1 \ldots Sn$, insert an attribute tag Sx–$IDREF$ of type $IDREF$.



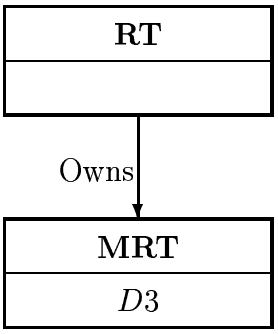

```
<complexType name="MRT-RecType">
    <sequence>
        <element name="D3" type="string" />
    </sequence>
    <attribute name="Ownes-IDREF" type="IDREF" />
</complexType>
```

STEP 5: The next step is to create complex types for the record types, which include zero to many records. For each record type RT, create a complex type RT–$RecTypeType$, and include an element RT–Rec of type RT–$RecType$. Set the $minOccurs$ attribute to 0 and $maxOccurs$ attribute to *unbounded* for each element.



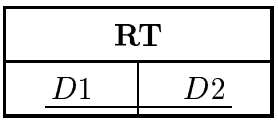

```
<complexType name="RT-RecTypeType">
    <sequence>
        <element name="RT" type="dbns:RecType"
            maxOccurs="0" minOccurs="unbounded" />
    </sequence>
</complexType>
```

STEP 6: In step four the database element is created. For the database DB with record types $RT1 \ldots RTn$, add DB element to schema and insert an anonymous complex type. Then, include $RT1$–$RecTypeldotsRTn$–$RecType$ as elements with corresponding complex types of $RT1$–$RecTypeType \ldots RTn$–$RecTypeType$.




```
<element name="DB">
    <complexType>
        <sequence>
            <element name="RT1"
                type="dbns:R1-RecTypeType" />
            <element name="RT2"
                type="dbns:R2-RelRecTypeType" />
        </sequence>
    </complexType>
</element>
```

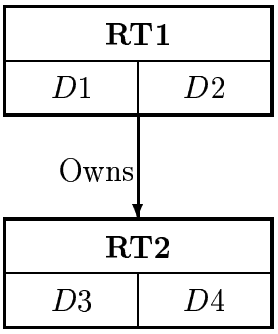
STEP 7: For each record type *RT*, if duplicates are not allowed for data items $D1 \ldots Dn$, insert a unique tag $RT\text{–}D1 \ldots Dn\text{–}DNA$ to the database element. Add selector tag with the *xpath* value set to $RT/RT\text{–}Rec$ and insert $D1 \ldots Dn$ field tags with *xpath* value set to $@D1 \ldots @Dn$.




```
<unique name="RT-D1_D2-DNA">
    <selector xpath="dbns:RT-RecType/dbns:RT" />
    <field xpath="@D1" />
    <field xpath="@D2" />
</unique>
```

3.2.1 Sample Network Database

The XML schema for a sample network data model is shown below.




```
<?xml version="1.0" encoding="UTF-8"?>
<schema
    targetNamespace="http://www.example-uri.com/Sample-DB"
    xmlns:dbns="http://www.example-uri.com/Sample-DB"
    xmlns="http://www.w3.org/2001/XMLSchema"
    elementFormDefault="unqualified"
attributeFormDefault="unqualified">

    <element name="Sample-DB">
        <complexType>
            <sequence>
                <element name="R1-Rel" type="dbns:R1-RelType" />
                <element name="R2-Rel" type="dbns:R2-RelType" />
            </sequence>
        </complexType>
        <key name="R1-Key">
            <selector xpath="dbna:R1-Rel/dbns:R1" />
            <field xpath="@A1" />
        </key>
        <key name="R2-Key">
            <selector xpath="dbns:R2-Rel/dbns:R2" />
                <field xpath="@A5" />
                <field xpath="@A6" />
        </key>
        <keyref name="R2-A5" refer="R1-Key">
            <selector xpath="dbns:R2-Rel/dbns:R2" />
            <field xpath="@A5" />
        </keyref>
    </element>
    <complexType name="R1-RelType">
        <sequence>
            <element name="R1" type="dbns:R1-RecType"
                minOccurs="0" maxOccurs="unbounded" />
        </sequence>
    </complexType>
    <complexType name="R2-RelType">
```

```
        <sequence>
            <element name="R2" type="dbns:R2-RecType"
                minOccurs="0" maxOccurs="unbounded" />
        </sequence>
    </complexType>
    <complexType name="R1-RecType">
        <sequence>
            <element name="A1" type="int" />
            <element name="A2" type="string" />
            <element name="A3" type="string" />
            <element name="A4" type="float" nillable="true" />
        </sequence>
    </complexType>
    <complexType name="R2-RecType">
        <sequence>
            <element name="A5" type="int" />
            <element name="A6" type="string" />
            <element name="A7" type="date" />
        </sequence>
    </complexType>
</schema>
```

4 libSyD: Library System of Databases

The relational to XML algorithm presented above has been implemented using relational databases as the data sources. The prototype, libSyD, attempts to mimic a public library system. The system is an interactive Web application that is comprised of eight different library branches and a query engine. The query engine acts as a single point of entry for the 8 branches. Users of the system may issue title and author queries using their favorite browser as the user interface. Data is stored in the relational database mySQL at every site. However, our algorithm appliesx to any relational database system with a JDBC driver that supports catalog access functions. No network database is included in the libSyD prototype.

4.1 libSyD Implementation

The architecture for libSyD (see Figure 1) is organized in three layers, the User Layer, the Query Engine Layer, and the Data Source Layer. Each layer utilizes XML to exchange data. In addition, XML schemas are proposed as the unifying data model for querying and integrating data from the heterogeneous data sources.

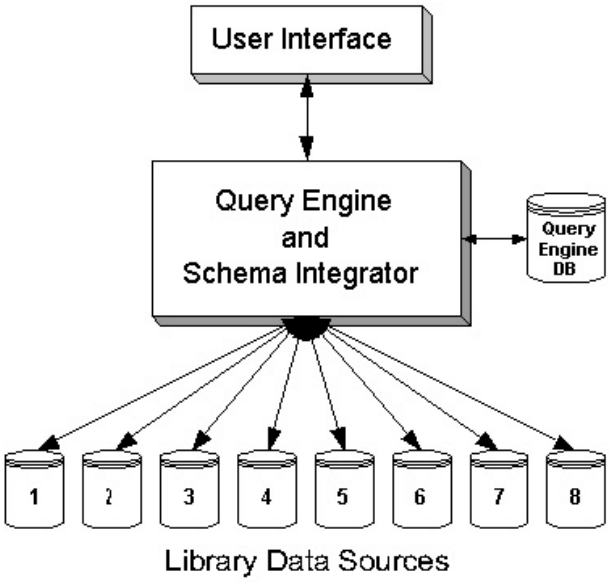


Fig. 1. Architecture

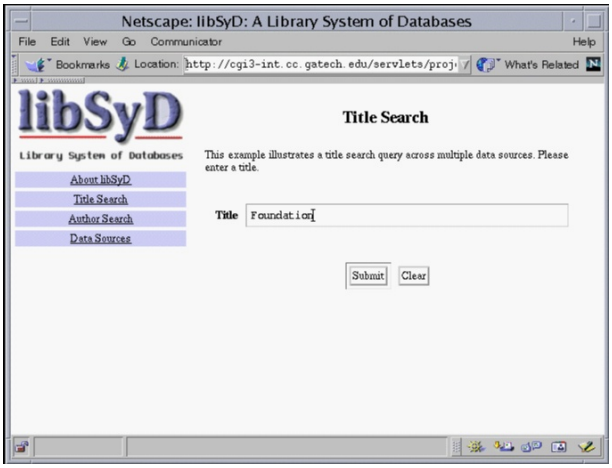


Fig. 2. Title Search Form

The user interface is a Web site which allows the user to initiate a title and author search (see Figure 2). This Web site can be accessed by any standard Web browser. Other applications that are directly connected to the Internet can also be developed as user interfaces to the query engine.

The user requests information through queries submitted to the system via an HTML form. Figure 2 shows an example of a query for books with the word "Foundation" found in the title.

Once the query has been submitted it is sent to the query engine in an XML document. Sub-query XML documents are then generated and sent to the data sources at the lower layer. The data sources translate the queries into equivalent SQL statements, execute the queries and return the results to the

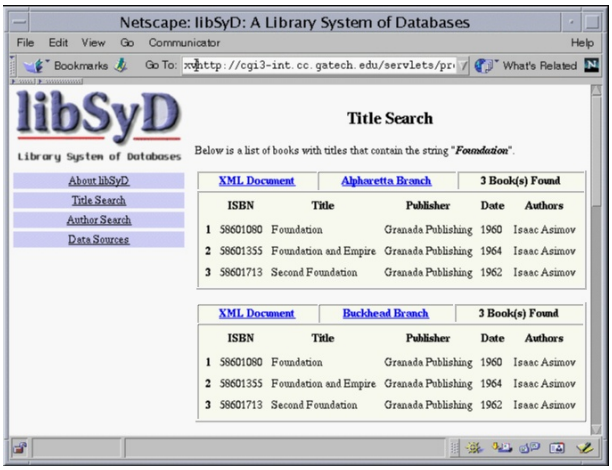


Fig. 3. Title Search Result

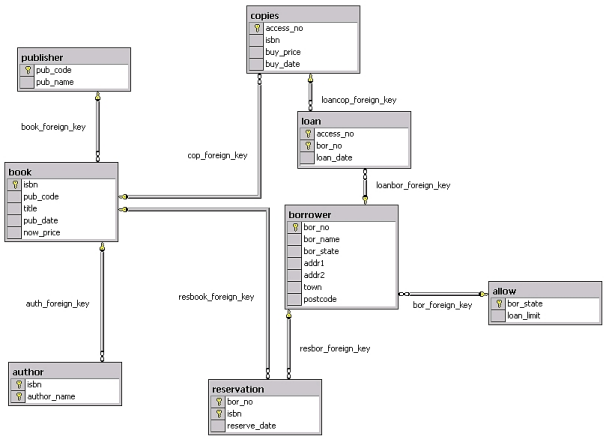


Fig. 4. Library Database Schema

query engine as XML documents. Once all the resulting XML documents are received, the query engine combines them and presents them to the user interface for viewing (Figure 3).

4.2 Data Sources

The key components of the system are the data sources at the Data Source Layer. The main function of a data source is to provide a common interface for the Query Engine Layer to each library database. In this implementation of libSyD they share the same schema (Figure 4). They are thus homogeneous, but independent. A data source's duties include:

(1) Map relational schema into XML Schema
(2) Parse and validate the XML query documents that are sent from the query engine

(3) Generate equivalent queries in SQL and execute the queries against the database
(4) Translate the results of the query into an XML document and return it to the query engine

4.3 libSyD Implementation

libSyD is an interactive Web application built using Java Servlets and JDBC. The system shown comprises eight different data sources representing libraries (Alpharetta, Buckhead, Central, Kirkwood, Northside, Peachtree, Roswell, Southwest) and a query engine database maintained in mySQL running on a Microsoft Windows NT 4.0 Server. The query engine database maintains information about the different data sources in the system. The XML parser for the system is Apache's Xerces parser. The query engine servlet and the data source servlets are hosted on an Apache Jserv server.

5 Future Work

In this paper a framework for querying and integrating data from heterogeneous data sources was outlined. The proposed framework uses the XML schema as the canonical data model for schema integration. In this paper we presented algorithms for mapping relational and network data models to XML schemas, and illustrated them with examples. Future work, currently done in the scope of the XMLApe project, will include:

(1) Additional methods to facilitate schema mapping for legacy systems such as the hierarchical model
(2) Methods for integration of XML Schemas of the data sources into a static global schema
(3) Development of a data model for XML with graphical notation
(4) XML query processing (and a query algebra)
(5) XML query optimization
(6) Development of quality metrics for "goodness" of XML document

References

[1] T. Bray, J. Paoli, C. Sperberg-McQueen, E. Maler, Extensible markup language(XML) 1.0 (second edition), W3C (October 2000).
URL `http://www.w3.org/TR/2000/REC-xml-20001006/`

[2] P. Biron, A. Malhotra, XML schema part 2: Datatypes; W3C recommendation, W3C (2001 May).
URL http://www.w3.org/TR/2001/REC-xmlschema-2-20010502/

[3] D. Fallside, XML schema part 0: Primer; W3C recommendation, W3C (May 2001).
URL http://www.w3.org/TR/2001/REC-xmlschema-0-20010502/

[4] H. Thompson, D. Beech, M. Maloney, N. Mendeldohn, XML schema part 1: Structures; W3C recommendation, W3C (May 2001).
URL http://www.w3.org/TR/2001/REC-xmlschema-1-20010502/

[5] S. Chung, P. Mah, Schema integration for multidatabases using the unified relational and object-oriented model, Proceedings of the 1995 ACM 23rd annual conference on computer science (1995) 208–215.

[6] M. Bright., A. Hurson, S. Pakzad, A taxonomy and current issues in multidatabase systems, IEEE Computer 25 (3) (1992) 50–60.

[7] R. Ahmed, P. D. Smedt, W. Du, W. Kent, M. Ketabchi, W. Litwin, A. Rafii, M. Shan, The pegasus heterogeneous multidatabase system, IEEE Computer 24 (12) (1991) 19–27.

[8] B. Bhaskar, C. Egyhazy, K. Triantis, The architecture of a heterogeneous distributed database management system: The distributed access view integrated database (david), Proceedings of the 1992 ACM annual conference on Communications (1992) 173–179.

[9] M. Templeton, D. B. et al., Mermaid: A front-end to distributed heterogeneous databases, In Proceedings of the IEEE 75 (5) (1987) 695–708.

[10] T. Lee, M. Chams, R. Nado, M. Siegel, S. Madnick, Information integration with attribution support for corporate profiles, Procedings of the eighth international conference on information knowledge management (1999) 423–429.

[11] L. Mark, N. Roussopoulos, T. Newsome, P. Laohapipattana, Incrementally maintained network -¿ relational database mapping, Software Practice & Experience 22 (12) (1992) 1099–1131.

[12] J. Larson, S. Navathe, R. Elmasri, A theory of attribute equivalence in databases with application to schema integration, IEEE Transactions on Software Engineering 15 (4) (1989) 449–463.

[13] S. Navathe, R. Elmasri, J. Larson, Integrating user views in database design, IEEE Computer 19 (1) (1986) 50–62.

[14] Savasere, S. Navathe, et al., On applying classification to schema integration, In Proceedings of IEEE 1st International Workshop on Interoperability in Multidatabase Systems (1991) 258–261.

[15] C. Knoblock, J. Ambite, Agents for information gathering, In J. Bradshaw, editor, Software Agents. AAAI/MIT Press (1997) 347–374.

[16] N. Ashish, C. Knoblock, Semi-automatic wrapper generation for internet information sources, Conference on Cooperative Information Systems (1997) 160–169.

[17] A. Sahuguet, F. Azavant, Building light-weight wrappers for legacy web data-sources using w4f, The VLDB Journal (1999) 738–741.

[18] T. Bray, D. Hollander, A. Layman, Namespaces in XML, W3C (January 1999). URL `http://www.w3.org/TR/1999/REC-xml-names-19990114/`

[19] R. Elmasri, S. Navathe, Fundamentals of Database Systems, Vol. 3, Addison Wesley, 2000.

[20] DBTG, Report of the CODASYL Data Base Task Group, ACM .

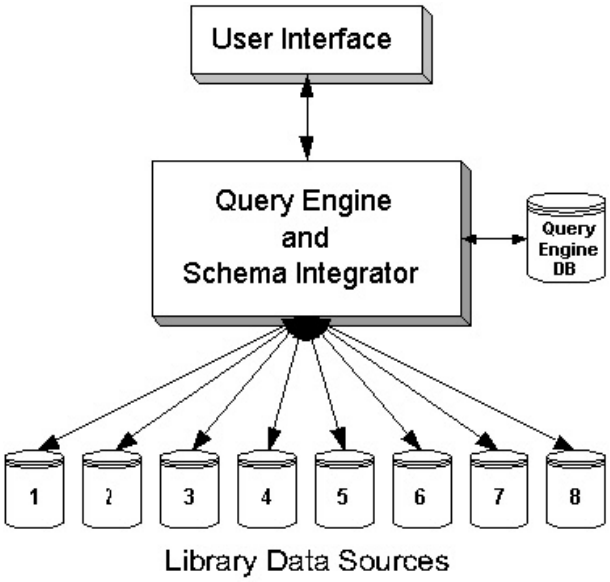


Fig. 5. Architecture

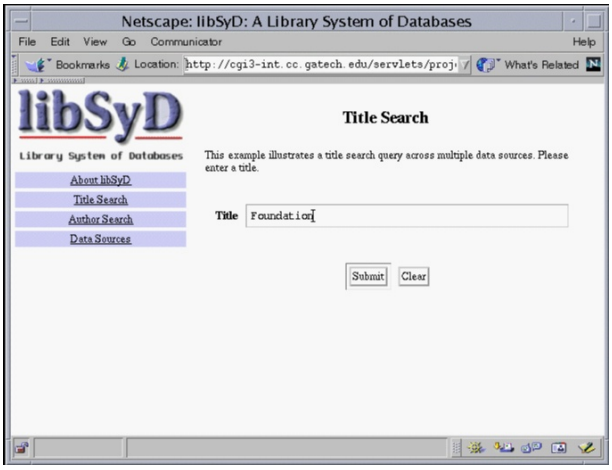


Fig. 6. Title Search Form

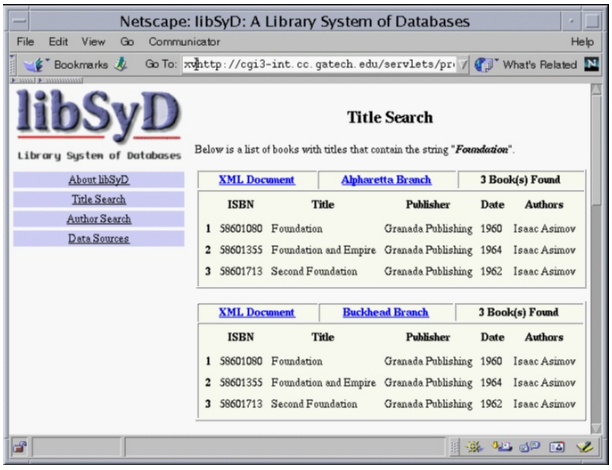


Fig. 7. Title Search Result

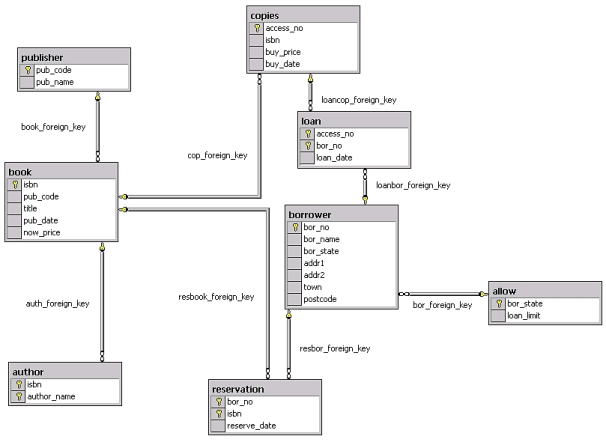


Fig. 8. Library Database Schema