Converting Legacy Relational Database into XML Database through Reverse Engineering

Chunyan Wang Anthony Lo Reda Alhajj Ken Barker
Advanced Database Systems and Applications Lab
Department of Computer Science
University of Calgary
Calgary, Alberta, CANADA
{*wangch, chiul, alhajj, barker*}*@cpsc.ucalgary.ca*

Key words: XML schema, relational schema, schema conversion, legacy database.

Abstract: XML (eXtensible Markup Language) has emerged and is being gradually accepted as the standard for data interchange over the Internet. Since most data is currently stored in relational database systems, the problem of converting relational data into XML assumes special significance. Many researchers have already done some accomplishments in this direction. They mainly focus on finding XML schema (e.g., DTD, XML-Schema, and RELAX) that best describes a given relational database with a corresponding well-defined database catalog that contains all information about tables, keys and constraints. However, not all existing databases can provide the required catalog information. Therefore, these applications do not work well for legacy relational database systems that were developed following the logical relational database design methodology, without being based on any commercial DBMS, and hence do not provide well-defined metadata files describing the database structure and constraints. In this paper, we address this issue by first applying the reverse engineering approach described in (Alhajj 2003) to extract the ER (Extended Entity Relationship) model from a legacy relational database, then convert the ER to XML Schema. The proposed approach is capable of reflecting the relational schema flexibility into XML schema by considering the mapping of binary and nary relationships. We have implemented a first prototype and the initial experimental results are very encouraging, demonstrating the applicability and effectiveness of the proposed approach.

1 Introduction

XML is emerging as the standard format for data exchange between different partners. Since most of the data nowadays reside in relational databases, it is important to automate the process of generating XML documents containing information from existing databases. Of course, one would like to preserve as much information as possible during the transformation process. The Relational-to-XML conversion involves mapping the relational tables and attributes names into XML elements and attributes names, creating XML hierarchies, and processing values in an application specific manner. Researcher mostly considered transforming relational databases that have rich corresponding catalogs. However, a large number of the existing relational databases are classified as legacy and the conversion of legacy relational databases has received little attention. Legacy databases are characterized by old-fashioned architecture, non-uniformity resulting from numerous extensions, and lack of the related documentation.

Realizing the importance of converting legacy databases into XML documents, we have developed a method that successfully handles the process. Our approach highly benefits from our previous finding on reverse engineering of legacy databases detailed in (Alhajj 2003). It leads to identifying and understanding all components of an existing database system and the relationships between them. Two basic steps are identified in the process of converting legacy databases into XML documents. First, reverse engineering is employed to deduce information about functional dependencies, keys and inclusion dependencies; the process involves reconstructing the ER model from an existing legacy database. Second, the obtained ER model is transformed into XML schema in a process known as forward engineering. Finally, our approach handles all type of relationships allowed in the ER model, including many-to-many and n-ary relationships.

The rest of the paper is organized as follows. Related work is discussed in Section 2. Section 3 is an overview of the reverse engineering process to extract ER model from the existing relational database; for more details, the reader is referred to (Alhajj 2003). ER model to XML schema conversion is presented in Section 4. A closer look at the developed approach and the implemented prototype is given in Section 5. Section 6 is the conclusions.

2 Related Work

There exist several tools that enable the composition of XML documents from relational data, such as IBM DB2 XML Extender, SilkRoute, and XPERANTO. XML Extender (Cheng and Xu 2000) serves as a repository for XML documents as well as their Document type declarations (DTDs), and also generate XML documents from existing data stored in relational database. It is used to define the mapping of DTD to relational tables and columns. XSLT and Xpath syntax are used to specify the transformation and the location path. SilkRoute (Fernandez, Tan and Suciu 2000) is described as a general, dynamic, and efficient tool for viewing and querying relational data in XML. XPERANTO (Carey et al 2000) is a middleware solution for publishing XML; object-relational data can be published as XML documents. It can be used by developers who prefer to work in a "pure XML" environment. However, the mapping from the relational schema to the XML schema is specified by human experts. Therefore, when a large relational schema and corresponding data need to be translated into XML documents, a significant investment of human effort is required to initially design the target schema. Finally, the work described in (Lee et al 2001) requires knowing the catalog contents in order to extract the relational database schema. The conversion of Relational-to-ER-to-XML has been proposed in (Fong, Pang and Bloor 2001). This reconstructs the semantic model, in the form of ER model, from the logical schema capturing user's knowledge, and then converts it to the XML document. However, many-to-many (M:N) and nary relationships are not considered properly. Finally, DB2XML (Turau 1999) is a tool for transforming data from relational databases into XML documents; DTDs are generated describing the characteristics of the data making the documents self contained and usable as a data exchange format.

Our approach is different from the above mentioned approaches; we focus on legacy relational databases. We adopt our reverse engineering approach proposed in (Alhajj 2003) to extract a semantically rich ER model from the given legacy relational database, and then we convert the ER model to XML schema; we consider M:N and n-ary relationships.

3 Extracting ER Model from Legacy Database

In this section, we present an overview of the reverse engineering process described in (Alhajj 2003). We will show the results obtained for the following running example.

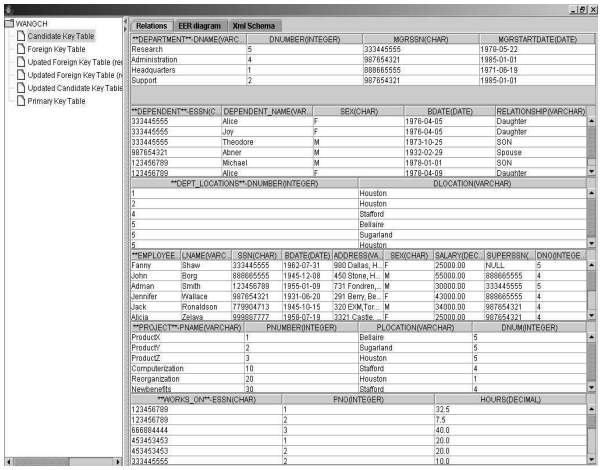
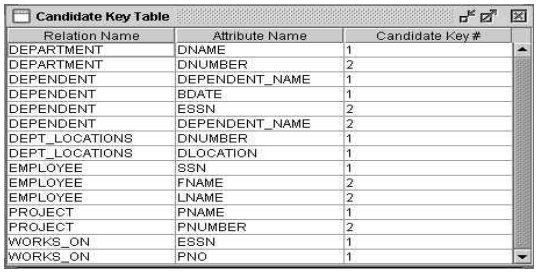


Figure 1: Example COMPANY relational database

Example 3.1 Consider the *COMPANY* database shown in Figure 1. This database contains six tables: *EMPLOYEE, DEPENDENT, PROJECT, DEPARTMENT, WORKS_ON*, and *DEPT_LOCATIONS*; and each table contains tuples some of which are shown in Figure 1.


Candidate Key Table

Relation Name	Attribute Name	Candidate Key #
DEPARTMENT	DNAME	1
DEPARTMENT	DNUMBER	2
DEPENDENT	DEPENDENT_NAME	1
DEPENDENT	BDATE	1
DEPENDENT	ESSN	2
DEPENDENT	DEPENDENT_NAME	2
DEPT_LOCATIONS	DNUMBER	1
DEPT_LOCATIONS	DLOCATION	1
EMPLOYEE	SSN	1
EMPLOYEE	FNAME	2
EMPLOYEE	LNAME	2
PROJECT	PNAME	1
PROJECT	PNUMBER	2
WORKS_ON	ESSN	1
WORKS_ON	PNO	1

Figure 2: Possible candidate Keys of all example relations

The first step is to extract all candidate and foreign keys found within the relations. The $CandidateKeys$ table shown in Figure 2 contains all possible candidate keys of the relations. The *Candidate Keys#* can be used to keep track of having the same attribute participating in more than one candidate key.

Foreign Key Table

CK Relation	CK Attribute	FK Relation	FK Attribute	Link number	CK Cardinality	FK Cardinality	Note
DEPARTMENT	DNUMBER	DEPT_LOCATIONS	DNUMBER	1	M >= 1	1	is not a candidate
DEPARTMENT	DNUMBER	EMPLOYEE	DNO	1	M >= 1	0 or 1	is not a candidate
DEPARTMENT	DNUMBER	PROJECT	DNUM	1	M >= 1	0 or 1	is not a candidate
EMPLOYEE	SSN	DEPARTMENT	MGRSSN	1	M >= 1	0 or 1	is not a candidate
EMPLOYEE	SSN	DEPENDENT	ESSN	1	M >= 1	0 or 1	is not a candidate
EMPLOYEE	SSN	EMPLOYEE	SUPERSSN	1	M >= 0	0 or 1	is not a candidate
EMPLOYEE	SSN	WORKS_ON	ESSN	1	M >= 1	0 or 1	is not a candidate
PROJECT	PNUMBER	WORKS_ON	PNO	1	M >= 1	1	is not a candidate
WORKS_ON	ESSN	DEPARTMENT	MGRSSN	1	M >= 1	0 or 1	is not a candidate
WORKS_ON	ESSN	DEPENDENT	ESSN	1	M >= 1	0 or 1	is not a candidate
WORKS_ON	ESSN	EMPLOYEE	SUPERSSN	1	M >= 0	0 or 1	is not a candidate
WORKS_ON	PNO	PROJECT	PNUMBER	1	1	0 or 1	is-a candidate

Figure 3: Foreign Keys and their corresponding candidate keys

The $ForeignKeys$ table shown in Figure 3 contains all pairs of attributes such that the first attribute is part of a candidate key in a certain relation and the second attribute is part of a foreign key, a representative of the first attribute within any of the relations. *Link#* is to differentiate different foreign keys in the same relation. Foreign keys are numbered so that all attributes within the same foreign key are assigned the same sequence number.

Primary Key Table

Relation Name	Attribute Name	Primary Key #
DEPARTMENT	DNUMBER	2
DEPENDENT	ESSN	2
DEPENDENT	DEPENDENT_NAME	2
DEPT_LOCATIONS	DNUMBER	1
DEPT_LOCATIONS	DLOCATION	1
EMPLOYEE	SSN	1
PROJECT	PNUMBER	2
WORKS_ON	ESSN	1
WORKS_ON	PNO	1

Figure 4: Primary keys for all the example relations

In general, a relation may have a set of candidate keys. One candidate key is chosen as the primary key by checking corresponding foreign keys. For relations that have multiple candidate keys, the primary key is selected to be the candidate key that appears in the first column of *ForeignKeys*. The $PrimaryKeys$ table for the example *COMPANY* database is shown in Figure 4.

The information in *ForeignKeys* is used in constructing what is called the *Relational Intermediate Directed (RID) Graph*, which present all possible unary and binary relationships between relations in the given relational schema. In the RID graph, each node represents a relation and two nodes are connected by a link to show that a foreign key in the relation that corresponds to the first node represents the primary key of the relation that corresponds to the second node.

As described in (Alhajj 2003), the cardinality of a relationship in the RID graph is determined as follows. A link is directed from R_2 to R_1 to reflect the presence of the primary key of R_1 as a foreign key in R_2; so, its cardinality is: *1:1* if and only if at most one tuple from R_2 holds the value of the primary key of a tuple from R_1; and *M:1* if more than one tuple from R_2 hold the value of the primary key of a tuple from R_1.

The employed process decides also on the minimum and maximum cardinalities at both sides of the link by investigating whether the link is optional or mandatory on each side.

Upated Foreign Key Table (removed symmetric reference)

CK Relation	CK Attribute	FK Relation	FK Attribute	Link number	CK Cardin...	FK Cardinality	Note
DEPARTMENT	DNUMBER	DEPT_LOCATIONS	DNUMBER	1	M >= 1	1	is not a candidate
DEPARTMENT	DNUMBER	EMPLOYEE	DNO	1	M >= 1	0 or 1	is not a candidate
DEPARTMENT	DNUMBER	PROJECT	DNUM	1	M >= 1	0 or 1	is not a candidate
EMPLOYEE	SSN	DEPARTMENT	MGRSSN	1	M >= 1	0 or 1	is not a candidate
EMPLOYEE	SSN	DEPENDENT	ESSN	1	M >= 1	0 or 1	is not a candidate
EMPLOYEE	SSN	EMPLOYEE	SUPERSSN	1	M >= 0	0 or 1	is not a candidate
EMPLOYEE	SSN	WORKS_ON	ESSN	1	M >= 1	0 or 1	is not a candidate
PROJECT	PNUMBER	WORKS_ON	PNO	1	M >= 1	1	is not a candidate
WORKS_ON	ESSN	DEPARTMENT	MGRSSN	1	M >= 1	0 or 1	is not a candidate
WORKS_ON	ESSN	DEPENDENT	ESSN	1	M >= 1	0 or 1	is not a candidate
WORKS_ON	ESSN	EMPLOYEE	SUPERSSN	1	M >= 0	0 or 1	is not a candidate

Figure 5: ForeignKeys after eliminating symmetry

Updated Foreign Key Table (removed transitive reference)

CK Relation	CK Attribute	FK Relation	FK Attribute	Link number	CK Cardinality	FK Cardinality	Note
DEPARTMENT	DNUMBER	DEPT_LOCATIONS	DNUMBER	1	M >= 1	1	is not a candidate
DEPARTMENT	DNUMBER	EMPLOYEE	DNO	1	M >= 1	0 or 1	is not a candidate
DEPARTMENT	DNUMBER	PROJECT	DNUM	1	M >= 1	0 or 1	is not a candidate
EMPLOYEE	SSN	DEPARTMENT	MGRSSN	1	M >= 1	0 or 1	is not a candidate
EMPLOYEE	SSN	DEPENDENT	ESSN	1	M >= 1	0 or 1	is not a candidate
EMPLOYEE	SSN	EMPLOYEE	SUPERSSN	1	M >= 0	0 or 1	is not a candidate
EMPLOYEE	SSN	WORKS_ON	ESSN	1	M >= 1	0 or 1	is not a candidate
PROJECT	PNUMBER	WORKS_ON	PNO	1	M >= 1	1	is not a candidate

Figure 6: ForeignKeys after eliminating transitivity

Analyzing the information in Figure 3, it can be easily observed that it contains some extra information because a foreign key is allowed to play the role of a candidate key and this leads to two *symmetric* and *transitive* references. Such extra information is deleted as described in (Alhajj 2003). The $CandidateKeys$ table for the example *COMPANY* database after deleting symmetric and transitive references are shown in Figures 5 and 6, respectively.

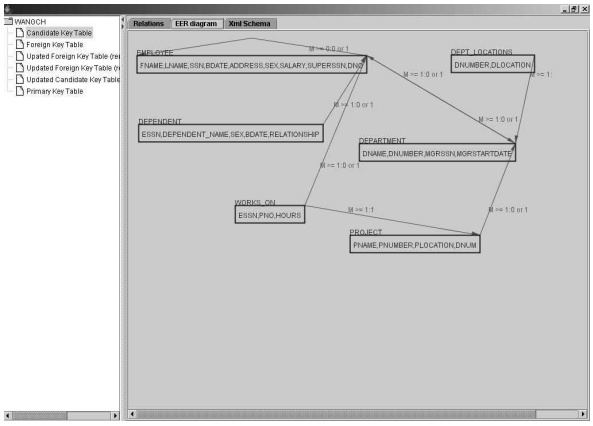


Figure 7: RID graph of the COMPANY database

Eliminating symmetric and transitive references lead to an optimized RID graph. The optimized RID graph is analyzed further to identify relationships with attributes, M:N and nary relationships, if any. The remaining unary and binary relationships are without attributes, and are represented by direct connections between nodes in the optimized RID graph. They are all classified as *1:1*, or *M:1*. The optimized RID graph for the example *COMPANY* database is shown in Figure 7.

4 Converting ER model to XML Schema

In this section, we present the proposed process for translating a conceptual schema (presented as RID graph) into XML schema. The process in pseudocode is depicted in Algorithm 4.1. **Algorithm 4.1 (ER Model to XML Schema Conversion)**
Input: The RID graph
Output: The corresponding XML schema
Step:

Step 1: Translate each entity in the ER model into a complex-type in XML schema.

Step 2: Map each attribute in every entity into a subelement within the corresponding complex-type.

Step 3: Create a root element and insert each entity in the ER model as a subelement with the corresponding complex-type.

Step 4: Use "key" and "keyref" to map each relationship between any two entities.

EndAlgorithm

In order to convert an ER model to XML schema by Algorithm 4.1, we need to go through the four steps as detailed next:

- Each entity E of the ER model is translated into an XML complex-type of the same name E in the XML schema. In each complex-type E, there is only one empty element. There will be several subelements inside the empty element. For example, the *PROJECT* entity is translated into a complex-type named "PROJECT_Relation". The empty element is called "PROJECT".

```
<complexType name="PROJECT_Relation" >
  <sequence>
    <element name="PROJECT"
    type="r:PROJECT_Type"
    maxOccurs="unbounded"/>
  </sequence>
</complexType>
<complexType name="PROJECT_Tuple">
  <sequence>
    . . . . . .
  </sequence>
</complexType>
```

  The cardinality constraint in the ER model can be explicated by associating two XML built-in *attributes*, also called indicators, namely "minOccurs" and "maxOccurs", with subelements under the XML complex-Type. The "maxOccurs" indicator specifies the maximum number of times a subelement can occur. "maxOccurs" = "unbounded" indicates the element may appear more than once. The "minOccurs" indicator specifies the minimum number of times a subelement can occur. The default value for both the "maxOccurs" and the "minOccurs" attributes is 1. If we want to specify a value only for "minOccurs", it must be either 0 or 1. Similarly, if we want to specify a value only for the "maxOccurs", it should be greater than or equal to 1. If both "minOccurs" and "maxOccurs" are omitted, then the subelement must appear exactly once.

- In step 2 of Algorithm 4.1, each attribute A_i of the entity E is mapped into a subelement of the corresponding complex-type E. For example, the *PROJECT* entity is mapped into a complex-type named "PROJECT_Tuple". Inside the "PROJECT_Tuple" complex-type, there are several subelements such as *PNAME*, *PNUMBER*, *PLOCATION*, and *DNUM*. They are the attributes of the "PROJECT" entity. The XML schema of the *PROJECT* entity is:

```
<complexType name = "PROJECT_Tuple">
  <sequence>
    <element ref="r:PNAME" />
    <element ref="r:PNUMBER" />
    <element ref="r:PLOCATION"/>
    <element ref="r:DNUM"/>
  </sequence>
</complexType>
```

  The <sequence> specification in the XML schema captures the sequential semantics of a set of subelements. For instance, in the <sequence> given above, the subelement *PNAME* comes first, followed by *PNUMBER*, and then *PLOCATION*, with *DNUM* at the end. These subelements must appear in instance documents in the same sequential order as they are declared here. XML schema also provides another constructor called <all>, which allows elements to appear in any order, and all the elements must appear once or not at all.

- In step 3 of Algorithm 4.1, each entity is mapped into the XML schema. We first need to create a root element that represents the entire given legacy relational database. We create the root element as a complex-type in the XML schema, and then insert each entity as a subelement of the root element. Next is an example which contains the six entity objects *DEPARTMENT, DEPENDENT, DEPT_LOCATIONS, EMPLOYEE, PROJECT, WORKS_ON*. We call the root element *COMPANY*:

```
<element name = "COMPANY">
  <complexType>
    <sequence>
      <element name="r:DEPARTMENT_Tpype" />
      <element name= "r:DEPENDENT_Type" />
      <element name="r:DEPT_LOCATIONS_Type"
      />
```

```
      <element name="r:EMPLOYEE_Type" />
      <element name="r:PROJECT_Type" />
      <element name="r:WORKS_ON_Type" />
    </sequence>
  </complexType>
</element>
```

Compared to DTD, the XML schema provides a more flexible and powerful mechanism through "key" and "keyref", which share the same syntax as "unique" and also make referential constraints possible in XML documents.

- In step 4 of Algorithm 4.1, we use the elements "key" and "keyref" to enforce the uniqueness and referential constraints among the data. According to (**?**), the "key" element specifies an attribute or element value as a key (unique, non-nullable, and always present) within the containing element in an instance document; and the "keyref" element specifies foreign keys, i.e., an attribute or element value correspond to that of an already specified key or unique element. The "key" and "keyref" elements replace and extent the capability of "ID", "IDREF" and "IDREFs" in DTD. They are among the great features introduced in XML schema. Also, we can use "key" and "keyref" to specify the uniqueness scope and multiple attributes to create the composite keys. Here is an example:

```
<key name = "PROJECTPrimaryKey">
  <selector xpath="r:PROJECT/r:PROJECT"/>
  <field xpath="PNUMBER"/>
</key>
<key name="WORKS_ON">
  <selector xpath="r:WORKS_ON/r:WORKS_ON"/>
  <field xpath="ESSN"/>
  <field xpath="PNO"/>
</key>
<keyref name = "PROJECTPNUMBER_WORKS_ONPNOReference"
 refer="r:PROJECTPrimaryKey">
  <selector xpath="r:WORKS_ON/r:WORKS_ON"/>
  <field xpath="PNUMBER"/>
</keyref>
```

In this example, we first specify the primary key for each entity in the ER model. From the *ForeignKeys* table, we know that *PNUMBER* is the primary key of *PROJECT* entity; *ESSN, and PNO* together form a composite primary key of *WORKS_ON* entity. *PNUMBER* is a foreign key of *WORKS_ON*, so we use *Keyref* to specify the foreign key relationship between *PROJECT* and *WORKS_ON* entities.

5 A Closer Look at the Developed approach

In this section, we describe the overall structure of our implementation. The purpose of this section is not to describe details of the code, but to grant the readers an overview of the system. We have two main components: extracting ER model from the given legacy relational database system and converting ER model to XML schema.

The prototype has been implemented using Java. In addition to the fact that we are familiar with Java, reasons for choosing Java include: 1) It is an object-oriented language, and hence it is easy to program in Java. 2) We can use JDBC driver to connect to the database, also there are some useful functions we can use for doing operations in the database. 3) We can use JDOM to obtain the XML schema.

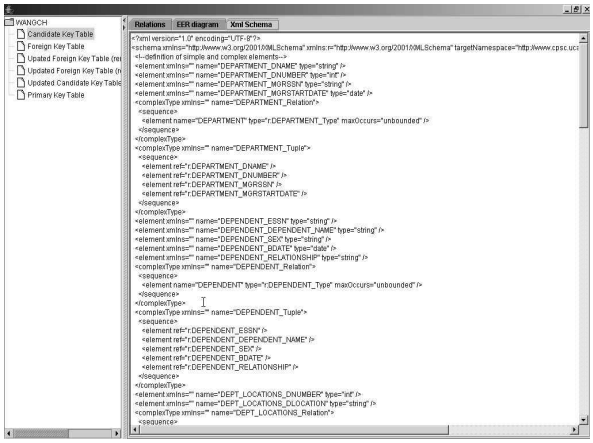


Figure 8: The output XML schema

We have tested our algorithms on the contents the *COMPANY* database in Example 3.1. The output XML schema for the *COMPANY* database is shown in Fig. 8. This supports the correctness, effectiveness and applicability of our approach.

We also test our approach on the contents of the *SAMPLE* database in DB2 and the *NorthWind* database in MS Access 2000, we neglected the catalog contents for both databases in order to test the reverse engineering process. It takes around 5 minutes for the *SAMPLE* database, and almost an hour for the *NorthWind* database. This is normal because we expect the time to increase when the size of the tested database increases. Compared to the *SAMPLE* database, it takes much longer time to test the *NorthWind*. The main reason is that *NorthWind* contains 8 tables, many attributes in some of tables, and a lot of records in each table. Most of the time is spent on analyzing the contents of the tables and deriving the ER model. Even if a human is asked to do the same job, the process becomes unmanageable manually as the size and complexity of the database increases.

To summarize, the proposed framework consists of the following major components to automatically extract the ER model from the given legacy relational

database, and then transfer it into XML schema.

Data layer, which is a legacy relational database that stores all the data to be analyzed and converted into XML.

Reverse engineering layer, which extracts an ER model from the input database.

Transformation layer, which transfers the ER model into XML schema.

Graphical output layer, which shows the result for each step (i.e., foreign keys table, candidate keys table, primary keys table, RID graph, and XML schema, etc).

Undoubtedly, reconstructing an ER model from a legacy database, and writing an XML schema file both are heavy and tedious jobs, especially for a large real application. The users could be relieved of this heavy load by using our framework. On the other hand, the users' knowledge could also be involved in this system. However, compared to reconstructing an ER model and writing a long XML schema file from scratch, the human's mental workload is greatly reduced with our framework.

Our framework presented in this paper has the following advantages compared to the work described in (Kleiner and Lipeck 2001), where the authors show how to obtain a DTD for data whose structure is described by a conceptual data model. In brief, they present the translation of all constructs of the ER model to DTDs and integrate them into an algorithm.

- Our framework could be used not only for a normal relational database system, but also for a legacy relational database system.
- We choose XML schema instead of DTD; XML schema provides a more flexible and powerful mechanism than DTD. We can easily present each entity in the ER model by using XML complex-Type. And also we can use "key" and "keyref" to declare the attributes uniqueness, composite keys, and referential constraints.
- Our prototype gives users a direct visualization of the output obtained from each phase of the process.
- The expected human workload is considerably reduced compared to the approach described in (Kleiner and Lipeck 2001).

6 Conclusions

In this paper, we presented a novel approach to extract an ER model from a legacy relational database, and then convert the ER model to a corresponding XML schema; i.e., by applying reverse engineering followed by forward engineering. We preserve as much information as we can from the given relational schema to the XML schema. Our approach not only works for commercial relational databases but also for legacy relational databases. We use the XML schema instead of the DTD schema; the advantages of this is that we can use a complex-type to represent each relational table; "key" and "keyref" are great features introduced in XML schema. They replace and extend the capability of "ID", and "IDREF" and "IDREFs" in DTD. We use "key" and "keyref" to specify the relationship between tables, the uniqueness scope and multiple attributes to create the composite keys. We can also determine *M:N* and *n*-ary relationships, so we produce a XML schemas and XML documents for the data stored in databases without knowing anything about the catalog information. Currently, we are working on improving the prototype to provide flexible visual querying facility by allowing the user to choose from the displayed RID graph the tables and even the attributes to be displayed in XML format.

REFERENCES

R. Alhajj, "Extracting the Extended Entity-Relationship Model from a legacy Relational Database, " *Information Systems,* Vol.28, No.6, pp.597-618, 2003.

M. Carey, et al, "XPERATO: Publishing Object-Relational Data as XML," *Proc. of the International Workshop on Web and Databases,* May 2000.

J. Cheng and J. Xu, *IBM DB2 XML Extender*, IBM Silcom Valley, February, 2000.

M.F. Fernandez, W.C. Tan, and D. Suciu, "SilkRoute: Trading between Relational and XML," *Proc. of the International Conference on World Wide Web*, May 2000.

J. Fong, F. Pang, and C. Bloor, "Converting Relational Database into XML Document," *Proc. of the International Workshop on Electronic Business Hubs,* pp61-65, Sep. 2001.

G. Kappel, et al, "X-Ray - Towards Integrating XML and Relational Database Systems," *Proc. of the International Conference on Conceptual Modeling,* pp. 339-353, Salt Lake City, UT, Oct. 2000.

C. Kleiner and U.W. Lipeck, "Automatic Generation of XML DTDs from Conceptual Database Schemas," *University of Hannover,* Germany, Sept 2001.

D. Lee, et al, "Nesting based Relational-to-XML Schema Translation," *Proc. of the International Workshop on Web and Databases,* May 2001.

M. Mani, D. Lee, and R. Muntz, "Semantic Data Modeling using XML Schemas," *Department of Computer Science, University of California, Los Angeles,* 2001.

V. Turau, "Making Legacy Data Accessible for XML applications," 1999, http://www.informatik.fh-wiesbaden.de/ turau/ps/legacy.pdf.