# An XML Storage System Using Object-Relational DBMSs: Syntax, Semantics, and Implementation

Wook-Shin Han[†], Ki-Hoon Lee[††], Byung S. Lee[†††], Won-Sik Kim[†]
[†] *Department of Computer Engineering, Kyungpook National University, South Korea*
[††] *Department of Computer Science and AITrc, KAIST, South Korea*
[†††] *Department of Computer Science, University of Vermont, U.S.A.*
*wshan@knu.ac.kr, khlee@mozart.kaist.ac.kr, bslee@emba.uvm.edu, wskim@www-db.knu.ac.kr*

**Abstract**

*As XML has become popular as a document standard in the World Wide Web, a lot of research has been done on the XML storage systems, which store and manage XML documents using existing DBMSs. In this paper, we present the syntax, semantics, and implementation of an XML storage system designed for an ORDBMS. Specifically, we first propose a specification language for describing the mapping using the XML Schema Language. Here, we describe the syntax and the semantics of the proposed language. Second, we propose an efficient algorithm for storing XML documents in an ORDB, based on the mapping information. We believe the proposed system is practically usable for object-relational DBMS implementors.*

*Keywords.* XML, ORDBMS

## 1. Introduction

XML is widely accepted as a new standard for documents having structural information on the Web [8]. Typically, the number of Web documents is very large and, therefore, storing and managing them require an efficient storage manager. There exist several types of XML storage systems, but most of them use relational DBMSs, and this is what currently available commercial DBMSs do as well [1, 4]. Naturally, their focus has been on storing XML documents as relational database records [2, 5, 6].

Storing XML documents as database records requires a specification of the mapping from the document structures to database schema. Currently, most commercial DBMSs provide such specification languages, but the languages are proprietary and limited to specifying a mapping to relational databases only. Such a limitation keeps us from exploiting the powerful modeling capabilities of object-relational DBMSs [7] like the references and the collections. Furthermore, learning the proprietary languages is a burden to users.

The contribution of this paper is as follows:

- we propose a specification language based on the standard XML Schema Language.
- we propose a SAX-based algorithm for storing XML documents in an OR database based on the mapping information.

The rest of the paper is as follows. Section 2 provides an overview of the XML supports in IBM DB2, a commercial DBMS. In Section 3, we propose an XML mapping language. In Section 4, we propose a detailed algorithm for storing XML documents. Section 5 concludes the paper.

## 2. XML SUPPORTS IN IBM DB2

IBM DB2 offers the XML Extender for storing XML documents. Its XML structure-to database schema mapping language is Document Access Definition (DAD). The DAD uses two elements, *element_node* and *attribute_node*, for describing the structure of XML documents, and one element *RDB_node* for describing its mapping to database schema. *Element_node* specifies the elements in an XML document, and *attribute_node* specifies the attributes. Both *element_node* and *attribute_node* have *RDB_node* as their subelement. *RDB_element* in turn has the subelements *table*, *column*, and *condition*, which respectively specifies a table, a column, and a primary key-foreign key relationship between tables in a database schema.

In the XML-to-schema mapping using DAD, XML elements are mapped to database tables or columns, XML attributes are mapped to database

columns, and relationships between XML elements are mapped to primary key-foreign key relationships between database tables. The principles for this mapping are as follows. First, specify the structure of XML documents by using *element_node* and *attribute_node*. Second, specify the mapping to a database schema using *RDB_node*. Third, specify all primary key-foreign key relationships between tables in the *RDB_node* subelement of the root *element_node*. Fourth, specify the table and column, to which an element or an attribute is mapped, in the *RDB_node* subelement of each non-root *element_node* and *attribute_node*.

Currently, most commercial DBMSs including DB2 have the following problems: 1) they provide proprietary mapping languages, and thus, learning the proprietary languages is a burden to users; 2) the mapping languages are limited to specifying a mapping to relational databases only. Such a limitation keeps us from exploiting the powerful modeling capabilities of object-relational (OR) DBMSs like the references and the collections.

## 3. XML MAPPING LANGUAGE FOR OBJECT-RELATIONAL DATABASES

To specify the explicit mappings from XML Schema to object-relational Schema, we need the following kinds of information: 1) information on the XML document structure, 2) information on the database schema, and 3) information on the mappings from the XML document structure to the database schema. We specify the XML document structure in the XML Schema Language, and specify the database schema and the mappings by adding three new subelements of the element *appinfo* in the XML Schema Language. Thus, we need only the XML Schema Language to specify all necessary mapping information.

The first two new subelements of the three are *Class* and *Column*. The element *Class* has the attribute *name* to specify the name of a class, and the element *Column* has the attributes *name* and *type* to specify the name and the type of a column, respectively. We use these two elements to specify the classes and columns in the database schema as well as the mappings from elements or attributes to classes or columns. Figure 1 shows the syntax for the *Class* and *Column* elements using XML Schma.

```
<xsd:element name = "Class">
    <xsd:complexType>
        <xsd:element ref="Column" minOccurs='1' maxOccurs='unbounded'/>
    </xml:complexType>
    <xsd:attribute name="name" type="xsd:string"/>
</xsd:element>
<xsd:element name = "Column">
    <xsd:complexType>
        <xsd:attribute name="name" type="xsd:string"/>
        <xsd:attribute name="type" type="xsd:string"/>
    </xsd:complexType>
</xsd:element>
```

**Fig. 1.** Syntax for the Class and Column elements.

The third new subelement is *Relationship*. It specifies the mapping from a relationship between two elements to a relationship between two classes. In an OO/OR database schema, the relationship between two classes can be represented using the reference type and the collection type.

The relationship is classified into one-to-one relationship and one-to-many relationship based on the cardinality constraint. It is also classified into a uni-directional relationship and a bi-directional relationship depending on whether the relationship exists in only one direction or in both directions.

To specify the mapping of a *relationship* as described above, the element Relationship has four attributes *parent*, *child*, *cardinality*, and *isOrdered*. The attribute *parent* specifies a reference-type column of the table mapped from a parent element, and the attribute *child* specifies a reference-type column of the table mapped from a child element. The attribute *cardinality* specifies the cardinality constraint of the relationship, and the attribute *isOrdered* specifies the order-preservation constraint of the relationship. As for the direction of a relationship, it is uni-directional if the attribute *child* is omitted, otherwise bi-directional. Figure 2 shows the syntax for the *Relationship* element using XML Schma.

```
<xsd:element name = "Relationship">
    <xsd:complexType>
        <xsd:attribute name="parent" type="xsd:string"/>
        <xsd:attribute name="child" type="xsd:string"/>
        <xsd:attribute name="cardinality" type="xsd:string"/>
        <xsd:attribute name="isOrdered" type="xsd:string"/>
    </xsd:complexType>
</xsd:element>
```

**Fig. 2.** Syntax for the Relationship element.

## 4. ALGORITHM FOR STORING XML DOCUMENTS IN AN O-R DATABASE

We use SAX API[3] to store very large XML documents into an ORDB. The SAX API incrementally parse XML documents with small memory, while DOM API should load the entire XML documents in main-memory before parsing them.

Figure 3 shows a SAX-style algorithm for storing XML documents in an ORDB. The algorithm *startElement* reads each XML element $E$ one by one and stores it in main-memory, and the algorithm *endElment* flushes in-memory objects into the database. In lines 1~16, if the element $E$ is mapped to a class, *startElement* creates an object $O$ of the class and forms a relationship with the object $O_p$ that stores the parent element. Specifically, if the relationship is one-to-one, in line 6 *startElement* stores the OID of $O$ in the column of $O_p$ that is a reference to $O$. If the relationship is one-to-many, in line 9 *startElement* stores the OID of $O$ in the column of $O_p$ that is a collection of references to $O$. Besides, if the relationship is bi-directional, in line 13 *startElement* stores the OID of $O_p$ in the column of $O$ that is a reference to $O_p$. In lines 17~21, if the element $E$ is mapped to a column, the value of $E$ is stored in the column. In lines 22-23, each attribute that belongs to the element $E$ is stored in the column of $O$ to which $E$ is mapped.

```
startElement(element E, attributes AL)
1: if (E is mapped to a class)
2:    /* create an in-memory object in the class
         to which E is mapped */
3:    O = CreateInmemoryObject(E);
4:    if (there exists an object O_p that stores
         the parent element and the relationship R)
5:       /* store the OID of O in the column of O_p
            that is reference to O */
6:       if (R is a one-to-one relationship)
7:          A = O_p .GetAttribute(R);
8:          A.Assign( GetOID(O));
9:       else if (R is a one-to-many relationship)
10:         /* store the OID of O in the column of O_p
               that is a collection of reference to O */
11:         A = O_p.GetAttribute(R)
12:         A.Insert(GetOID(O))
13:      if (R is a bidirectional relationship)
14:         /* store OID of O_p in the column of O
               that is a reference to O_p */
15:         A = O.GetAttribute(R)
16:         A.Insert(GetOID(O_p))
17: if (E is mapped to a column C)
18:    if (the type of C is simple)
19:       store the value of E in the column
          of the object O to which E is mapped
20:    else if (the type of C is a simple collection)
21:       store the value of E in the column of the
          object O to which E is mapped as a
          collection element;
22: for each attribute A in AL
23:    store the value of in the column of the
       in-memory object O to which E is mapped

endElement(element E)
1: FlushObject(O)
   /* flush the in-memory object O to which E is
      mapped into disk*/
2: DeleteObject(O)
   /* delete the in-memory object O*/
```

**Fig. 3.** A SAX-style algorithm for storing XML documents in an OR database.

## 5. CONCLUSION

In this paper, we proposed an XML storage system for storing and managing XML documents efficiently in an object-relational (OR) database. The system offers an XML-to-database mapping language based on the standard XML Schema Language, and provides methods usable in an ORDBMS as well as a relational DBMS. Specifically, we presented (1) a mapping specification language based on the results of the analysis, and (2) a SAX-style algorithm for storing XML documents in an OR database.

**Acknowledgment** This work was supported by Korea Research Foundation Grant (KRF-2003-003-D00347).

## References

[1] Cheng, J. and Xu J.: "XML and DB2," In Proc. the 16th Int'l Conf. on Data Engineering, pp. 569—573, San Diego, California, USA, 2000.

[2] Florescu, D. and Kossmann, D.: A Performance Evaluation of Alternative Mapping Schemes for Storing XML Data in a Relational Database, Technical Report RR-3680, INRIA, May 1999.

[3] Megginson, D., Simple API for XML (SAX), May 2000 (available from http://sax.sourcefourge.net).

[4] Microsoft Corp.: Microsoft SQL Server 2000, 2000.

[5] Shanmugasundaram, J. et al.: "Relational Databases for Querying XML Documents: Limitations and Opportunities," In Proc. 25th Int'l Conf. on Very Large Data Bases, pp. 302—314, Edinburgh, Scotland, UK, Sept. 1999.

[6] Shanmugasundaram, J. et al.: "A General Technique for Querying XML Documents Using a Relational Database System," ACM SIGMOD RECORD, Vol. 30, No. 3, Sept. 2001.

[7] Stonebraker, M. and Moore, D.: Object-Relational DBMSs: The Next Great Wave, Morgan Kaufmann, 1999.

[8] Simon, H., Strategic Analysis of XML for Web Application Development}, Computer Research Corp., 2000.