JOURNAL OF OBJECT TECHNOLOGY

Online at www.jot.fm. Published by ETH Zurich, Chair of Software Engineering ©**JOT, 2003**

Vol. 2, No. 3, May-June 2003

An XML Storage System for Object-Oriented/Object-Relational DBMSs

Wook-Shin Han
Department of Computer Engineering, Kyungpook National University, Korea
Ki-Hoon Lee
Department of Computer Science and AITrc, KAIST, Korea
Byung Suk Lee
Department of Computer Science, University of Vermont, U.S.A.

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 for storing and managing XML documents using existing DBMSs. Most of the research activities, however, assume a relational DBMS instead of an object-oriented/object-relational (OO/OR) DBMS, which offers more powerful modeling capabilities. In this paper, we present the design and implementation of an XML storage system designed for an OO/OR DBMS. Specifically, we first analyze the mapping from an XML document structure to OO/OR database schema. Second, we propose a method for describing the mapping using a standard language called the XML Schema Language. Third, we propose system catalog classes for storing the mapping information specified by users in the database. Fourth, we propose a detailed algorithm for storing XML documents in an OO/OR database, based on the mapping information. We believe the proposed system is practically usable for object-oriented programmers and DBMS implementors.

1 INTRODUCTION

XML is widely accepted as a new standard for documents having structural information on the Web [Sim00]. 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 [Che00, Mic00]. Naturally, their focus has been on storing XML documents as relational database records [Flo99, Sha99, Sha01].

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

Cite this article as follows: Woo-Shin Han, Ki-Hoon Lee, Byung Suk Lee: "An XML Storage System for Object-Oriented/Object-Relational DBMSs", in *Journal of Object Technology*, vol. 2, no. 3, May-June 2003, pp. 113-126. http://www.jot.fm/issues/issue_2003_05/article2

specifying a mapping to relational databases only. This limitation keeps us from exploiting the powerful modeling capabilities of object-oriented/object-relational (OO/OR) DBMSs [Sto99] like the references and the collections. Furthermore, learning the proprietary languages is a burden to users.

In this paper, we propose an XML storage system geared for an OO/OR DBMS. For this purpose, we first analyze the mapping from XML document structures to OO/OR schema and propose a specification language based on the standard XML Schema Language. Then, we propose a set of system catalog classes for storing user-specified mapping information in the database and propose a detailed algorithm for storing XML documents in an OO/OR database based on the mapping information.

The rest of the paper is as follows. Section 2 provides an overview of the XML Schema Language, and Section 3 provides an overview of the XML supports available from commercial DBMSs. Section 4 analyzes the mapping from the structure of XML documents to an OO/OR database schema and proposes an XML mapping language based on the analysis results. Section 5 proposes the system catalog classes and the detailed algorithm for storing XML documents. Section 6 concludes the paper.

2 XML SCHEMA LANGUAGE

The XML Schema Language is a standard from W3C that replaces XML Document Type Definitions (DTDs) for specifying the structure of an XML document [Bir01, Fal01, Tho01]. It is written in XML and offers several important elements including *xsd:element*, *xsd:attribute*, *xsd:complexType*, and *xsd:annotation*.

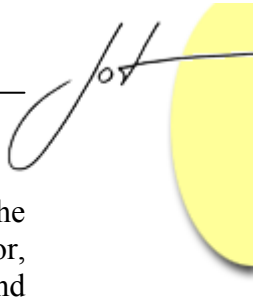
The element *xsd:element* is used for defining an element. It has the attributes *name* and *type* that respectively represent the name and type of the given element. Additionally it has the attributes *minOccurs* and *maxOccurs* that respectively represent the maximum and the minimum numbers of occurrences of the element.

The element *xsd:attribute* is used for defining an attribute. It has the attributes *name* and *type* that respectively represent the name and type of the given attribute.

The element *xsd:complexType* is used for defining the type of an element having subelements or attributes. In the XML Schema Language, if an element has subelements or attributes, the type of the element is called the complex type (complexType), and otherwise called the simple type.

The element *xsd:annotation* is used for annotating additional information like the comment on a document and the information used by application programs. It has two subelements—*xsd:documentation* and *xsd:appinfo*. The former is used for specifying comment and the latter is used for specifying information for application programs.

Figure 1 represents an exemplary XML Schema. The *xsd:element* with the optional *xsd:attribute* defines the elements *book*, *title*, *author*, etc. The element *book* has the subelements *title* and *author* representing the title and authors, respectively. It also has



the attribute *id* representing the unique id of a book. The element *author* has the subelements *name* and *email* representing the name and email address of the author, respectively. The *xsd:complexType* defines the complex types of the elements *book* and *author*. The *xsd:annotation* defines the annotation like the copyright information.

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:annotation>
    <xsd:documentation>
      Book schema for www.book.com.
      Copyright 2001 www.book.com. All rights reserved.
    </xsd:documentation>
  </xsd:annotation>
  <xsd:element name="book">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="title" type="xsd:string"/>
        <xsd:element name="author" minOccurs="0" maxOccurs="unbounded">
          <xsd:complexType>
            <xsd:sequence>
              <xsd:element name="name" type="xsd:string"/>
              <xsd:element name="email" type="xsd:string"/>
            </xsd:sequence>
          </xsd:complexType>
        </xsd:element>
      </xsd:sequence>
      <xsd:attribute name="id" type="xsd:integer"/>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

Fig. 1: An example of XML schema.

3 XML SUPPORTS IN COMMERCIAL DBMSS

In this section we give an overview of the XML structure-to-database schema mapping specification languages currently available from two selected commercial DBMSs: IBM DB2 and Microsoft SQL Server.

Note that the specification languages of both DBMSs are geared for a mapping to relational schema only and, therefore, cannot utilize such constructs as the references and collections available from an OO/OR schema. Furthermore, both languages are

proprietary to the specific DBMSs and, therefore, make it involving for users to switch between them.

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_n*ode.

Figure 3 shows a DAD specifying the mapping from the XML document structure in Figure 1 to the database schema in Figure 2. The table *book*, the table *author* and the primary key-foreign key relationship between the two tables are specified in the *RDB_node* subelement of the root *element_node* named “book.” The columns *id* and *title* of the table *book* are specified in the *RDB_node* subelements of the *attribute_node* named “id” and the *element_node* named “title,” respectively. Likewise, the columns *name* and *email* of the table *author* are specified in the *RDB_node* subelements of the *element_node* named “name” and the *element_node* named “email.”

Table book

name	id	title
type	integer	varchar(100)

Table author

name	bookId	name	email
type	integer	varchar(100)	varchar(100)

Fig. 2: An example relational database schema.

```
<element_node name="book">
    <RDB_node>
        <table name="book" key="id"/>                                  ] mapping the relationship between
        <table name="author"/>                                         ] the elements book and author
        <condition> book.id=author.bookId </condition>                 ]
    </RDB_node>
    <attribute_node name="id">
        <RDB_node>
            <table name="book"/>                                       ] mapping the attribute id
            <column name="id" type="integer"/>                         ]
        </RDB_node>
    </attribute_node>
    <element_node name="title">
        <text_node>
            <RDB_node>
                <table name="book"/>                                   ] mapping the attribute title
                <column name="title" type="varchar(100)"/>             ]
            </RDB_node>
        </text_node>
    </element_node>
    <element_node name="author" multi_occurrence="YES">
        <element_node name="name">
            <text_node>
                <RDB_node>
                    <table name="author"/>                             ] mapping the element author
                    <column name="name" type="varchar(100)"/>          ]
                </RDB_node>
            </text_node>
        </element_node>
        <element_node name="email">
            <text_node>
                <RDB_node>
                    <table name="author"/>                             ] mapping the element email
                    <column name="email" type="varchar(100)"/>         ]
                </RDB_node>
            </text_node>
        </element_node>
    </element_node>
</element_node>
```

Fig. 3: An example DAD.

Microsoft SQL Server

Microsoft SQL Server offers the XML Bulk Load utility for storing XML documents[Mic00]. Its XML-to-database mapping language is the annotated XML-Data Reduced (XDR) Schema. This language uses four elements—*ElementType, AttributeType, element, attribute*—for specifying the structure of an XML document, and two attributes *relation* and *field* as well as one element *relationship* for specifying the mapping to a database schema. Specifically, *ElementType* and *AttributeType* are used respectively to declare XML elements and attributes, whereas *element* and *attribute* are used to refer to the declared *ElementType* and *AttributeType*. Additionally, the attributes *relation* and

field respectively specify a table and a column, and the element *relationship* specifies the primary key-foreign key relationship between two tables in the database schema.

Like the DAD, the annotated XDR Schema maps XML elements and attributes to database tables or columns, and relationships between XML elements to primary key-foreign key relationships between database tables. The principles for specifying the mapping are like those for the DAD. First, specify the structure of XML documents by using the elements *ElementType*, *AttributeType*, *element*, and *attribute*. Second, specify the table and column, to which an element or an attribute is mapped, by using the attributes *relation* and *field*. Third, specify the primary key-foreign key relationship between tables by using the element *relationship*.

Figure 4 shows an annotated XDR Schema corresponding to the DAD in Figure 3. The element *book* is mapped to the table *book*, and the attribute *author* and the element *title* are respectively mapped to the columns *id* and *title* of the table *book*. Likewise, the element *author* is mapped to the table *author*, and the attribute *name* and the element *email* are respectively mapped to the columns *name* and *email* of the table *author*. Besides, the relationship between the two elements *book* and *author* is mapped to the primary key-foreign key relationship between the two tables *book* and *author*.

```
<Schema xmlns="urn:schemas-microsoft-com:xml-data"
        xmlns:dt="urn:schemas-microsoft-com:datatypes"
        xmlns:sql="urn:schemas-microsoft-com:xml-sql">

    <ElementType name="book" sql:relation="book">                          ] mapping the elements
        <element type="title" sql:field="title"/>                          ]   book and title
        <element type="author" minOccurs="0" maxOccurs="*">
            <sql:relationship key-relation="book" key="id"                 ] mapping the relationship between
                              foreign-relation="author" foreign-key="bookId"/>  ]   the elements book and author
        </element>
        <attribute type="id" sql:field="id"/>                              ] mapping the attribute id
    </ElementType>

    <ElementType name="title" dt:type="string"/>
    <AttributeType name="id" dt:type="int"/>

    <ElementType name="author" sql:relation="author">                      ]
        <element type="name" sql:field="name"/>                            ] mapping the elements author,
        <element type="email" sql:field="email"/>                          ]   name, and email
    </ElementType>

    <ElementType name="name" dt:type="string"/>
    <ElementType name="email" dt:type="string"/>
</Schema>
```

Fig. 4: An example of annotated XDR schema.

4 DESIGN OF AN XML MAPPING LANGUAGE

In this section we analyze the mappings from an XML document structure to a database schema and, based on the analysis results, propose an XML structure-to-database schema mapping language designed for OO/OR databases.

Analysis of Mapping from XML Document Structure to Database Schema

Figure 5 shows possible mappings from an XML document structure to an OO/OR schema. The rectangle on the left side shows the components of an XML document, and the rectangle on the right hand side shows the components of an OO/OR database schema. The arrows denote possible mappings between the two. As shown in the arrows numbered 1 through 4, each XML element or attribute can be mapped to either a database class or a column. However, if XML attributes are mapped to classes, join operations are required unnecessarily when processing queries and, therefore, we do not allow this kind of mapping in this paper. The arrows for these disallowed mappings are distinguished with broken lines in Figure 5.

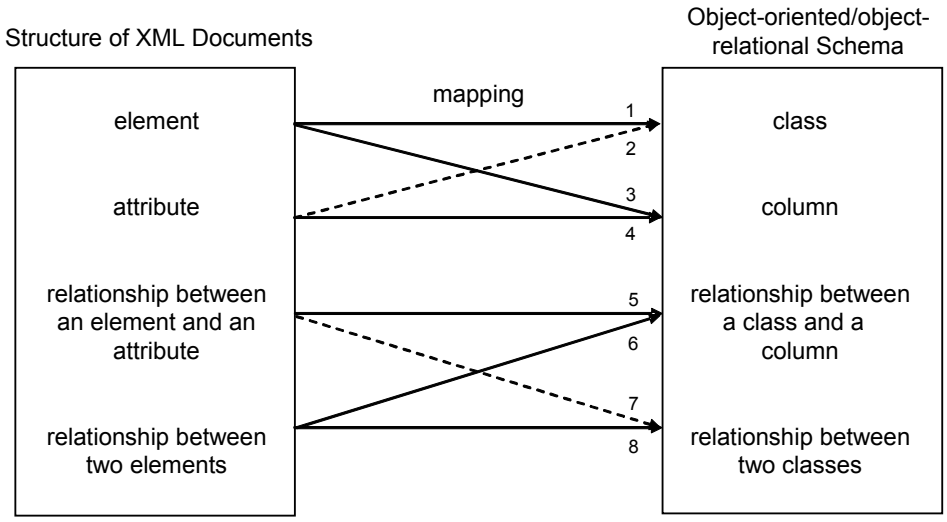


Fig. 5: Possible mappings from an XML structure to an object-oriented/object-relational schema.

A relationship between an element and an attribute can be mapped to either a relationship between a class and a column (i.e., arrow 5) or a relationship between a class and another class (i.e., arrow 7). However, only the latter option applies because XML attributes are mapped to only the database columns. The relationship between an element and another element is mapped to either the relationship between a class and a column (i.e., arrow 6) or the relationship between two classes (i.e., arrow 8).

We do not need to explicitly specify the mappings to a relationship between a class and a column (i.e., arrows 5 and 6) because this relationship is already maintained in the database schema. That is, as long as we know the mappings between XML

elements/attributes and database classes/columns (i.e., arrows 1, 2, and 4), we can find a relationship between an XML element and an attribute or between two elements that is mapped to to the relationship between a class and a column.

XML Mapping Language for OO/OR Databases

To specify the explicit mappings (i.e., arrows 1, 3, 4, and 8), 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.

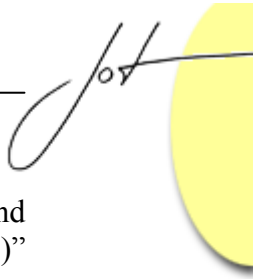
The first two new subelements of the three are *Class* and *Column*. The element *Class* has the attribute *name* to specify the name of a class, and the element *Column* has the attributes *name* and *type* to specify the name and the type of a column, respectively. We use these two elements to specify the classes and columns in the database schema as well as the mappings from elements or attributes to classes or columns (i.e., arrows 1, 3, and 4).

The third new subelement is *Relationship.* It specifies the mapping from a relationship between two elements to a relationship between two classes (i.e., arrow 8). 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. The direction of a relationship is uni-directional if the attribute *child* is omitted, otherwise bi-directional.

Figure 6 shows an example of a mapping written in this proposed specification language. It shows a portion of mapping an XML document structure in Figure 1 to an OO/OR schema in Figure 7. The mapping information is annotated with subelements of the element *appinfo*. Specifically, the database schema is specified with the elements *Class* and *Column* on their own, and the mapping from the elements or attributes to the database schema is specified with the two elements used as subelements of the elements or attributes. The relationship between the class *book* and the class *author* is one-to-many



and ordered as specified with the attributes '*cardinality*="onetoMany"' and '*isOrdered*="yes"' of the *Relationship* element. Here, use the list type, "list(ref(author))" instead of the set type to preserve the order of the relationship.

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:annotation>
    <xsd:appinfo>
      <Class name="book">                                          ┐
        <Column name="book.authors" type="list(ref(author))"/>     │
        <Column name="book.id" type="integer"/>                    │ information on the
        <Column name="book.title" type="varchar(100)"/>            │ database schema
      </Class>                                                     │
      <Class name="author">                                        │
        <Column name="author.book" type="ref(book)"/>              │
        <Column name="author.name" type="varchar(100)"/>           │
        <Column name="author.email" type="varchar(100)"/>          ┘
      </Class>
      <Relationship parent="book.authors" child="author.book"      ┐ mapping the relationship
                    cardinality="onetoMany" isOrdered="yes"/>      ┘ between the elements
    </xsd:appinfo>                                                   book and author
  </xsd:annotation>
  <xsd:element name="book">
    <xsd:annotation>
      <xsd:appinfo>                                                ┐ mapping the element
        <Class name="book"/>                                       ┘ book
      </xsd:appinfo>
    </xsd:annotation>
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="title" type="xsd:string">
          <xsd:annotation>
            <xsd:appinfo>                                          ┐ mapping the element
              <Column name="book.title"/>                          ┘ title
            </xsd:appinfo>
          </xsd:annotation>
          . . .
```

Fig. 6: An example mapping from the structure of XML documents to object-oriented/object-relational schema.

Class book

name	id	title	authors
type	integer	varchar(100)	list(ref(author))

Class author

name	book	name	email
type	ref(book)	varchar(100)	varchar(100)

Fig. 7: An example OO/OR database schema for Figure 2.

5 STORING XML DOCUMENTS BASED ON THE MAPPING

In this section we describe a set of database system catalog classes for storing user-provided XML structure-to-database schema mapping information, and present an algorithm for storing XML documents in an OO/OR database according to the mapping information.

Catalog Classes for Storing the Mapping information

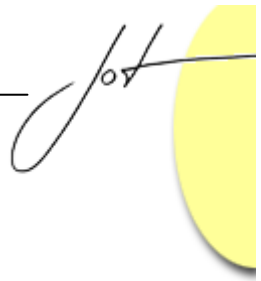
As mentioned in Section 4, we need to store the information on the XML document structure, the database schema, and the mapping between them. Since the information on the database schema is already stored in the database system catalog, we have only to store the information on the XML document structure and the mapping.

In this paper we store the information on XML document structures and the information on mappings in the same classes to avoid unnecessary joins. Figure 8 shows the catalog classes used for this purpose. The class *xmlSysElements* is for each element and its mapping, the class *xmlSysAttributes* is for each attribute and its mapping, and the class *xmlSysRelationships* is for each relationship (between two XML elements) and its mapping.

The class *xmlSysElements* has the columns *elementId* and *elementName* for storing an element. Additionally, it has the columns *flag*, *classId*, and *columnNo* for storing the mapping information of the element. Specifically, the column *flag* specifies whether the element is mapped to a class or a column, the column *classId* specifies the identifier of the class the element is mapped to, and *columnNo* specifies the number of the column the element is mapped to.

The class *xmlSysAttributes* has the columns *elementId* and *attributeNo*, and *attributeName* for storing an attibute. Additionally, it has the columns *classId* for storing the identifier of the class the attribute is mapped to and the column *columnNo* for storing the number of the column the attribute is mapped to.

The class *xmlSysRelationships* has the columns *parentId*, *childId*, and *cardinality* for storing a relationship. Additionally, it has the columns *flag*, *isOrdered*, *parentClassId*, *parentColumnNo*, *childClassId*, and *childColumnNo* for storing the mapping information of the relationship. Specifically, the column *flag* specifies whether the child element is mapped to a class or to a column of the class mapped from the parent element, the columns *parentClassId* and *parentColumnNo* specify the class and the column the parent element is mapped to, and the columns *childClassId* and *childColumnNo* specify those the child element is mapped to.



xmlSysElements

name	elementId	elementName	flag	classId	columnNo
type	integer	varchar	char	integer	integer

xmlSysAttributes

name	elementId	attributeNo	attributeName	classId	columnNo
type	integer	integer	varchar	integer	integer

xmlSysRelationships

name	parentId	childId	cardinality	flag	isOrdered	parentClassId	parentColumnNo	childClassId	childColumnNo
type	integer	integer	char	char	char	integer	integer	integer	integer

Fig. 8: Catalog classes for storing the mapping information.

Algorithm for Storing XML documents in an OO/OR database

Figure 9 shows the algorithm StoreXML_ORDB for storing XML documents in an OO/OR database. The algorithm reads each XML element E one by one and stores it in the database. In lines 3-15, if the element E is mapped to a class, the algorithm 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 9 the algorithm 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 11 the algorithm 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 the algorithm stores the OID of O_p in the column of O that is a reference to O_p. In lines 16-22, if the element E is mapped to a column, the value of E is stored in the column. In lines 23-24, each attribute that belongs to the element E is stored in the column of O to which E is mapped.

6 CONCLUSION

In this paper, we proposed an XML storage system for storing and managing XML documents efficiently in an object-oriented/object-relational (OO/OR) database. The system offers an XML-to-database mapping language based on the standard XML Schema Language, and provides methods usable in an OO/OR DBMS as well as a relational DBMS. Specifically, we analyzed the mapping from an XML document structure to an OO/OR database schema, and presented (1) a mapping specification language based on the results of the analysis, (2) database catalog classes for storing user-provided mapping information, and (3) an algorithm for mapping and storing XML documents in an OO/OR database.

```
Algorithm StoreXML_ORDB
Input:
  D : XML document

1:  for (element E in the XML document D)
2:  {
3:     if (E is mapped to a class)
4:     {
5:        create an object O in the class to which E is mapped;
6:        if  (there exists an object O_p that stores the parent element and the relationship R)
7:        {
8:           if (R is a one-to-one relationship)
9:              store the OID of O in the column of O_p that is a reference to O;
10:          else if (R is a one-to-many relationship)
11:             store the OID of O in the column of O_p that is a collection of references to O;
12:          if (R is a bi-directional relationship)
13:             store the OID of O_p in the column of O that is a reference to O_p;
14:       }
15:    }
16:    else if (E is mapped to a column C)
17:    {
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:    }
23:    for (each attribute A that belongs to E)
24:       store the value of A in the column of the object O to which E is mapped;
25: }
```

Fig. 9: An algorithm for storing XML documents in an OO/OR database.

ACKNOWLEDGEMENT

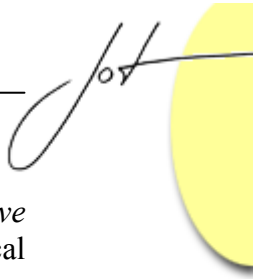
This work was supported by the Korea Science and Engineering Foundation(KOSEF) through the Advanced Information Technology Research Center(AITrc).

REFERENCES

[Bir01] Biron, P. and Malhotra, A.: XML Schema Part 2: Datatypes, May 2001 (available from http://www.w3.org/TR/xmlschema-2).

[Che00] 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.

[Fal01] Fallside, D.: XML Schema Part 0: Primer, May 2001 (available from http://www.w3.org/TR/xmlschema-0).



[Flo99] 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.

[Mic00] Microsoft Corp.: Microsoft SQL Server 2000, 2000 (available from http://www.microsoft.com/sql/default.asp).

[Sha99] 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.

[Sha01] 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.

[Sto99] Stonebraker, M. and Moore, D.: *Object-Relational DBMSs: The Next Great Wave*, Morgan Kaufmann, 1999.

[Sim00] Simon, H., *Strategic Analysis of XML for Web Application Development*, Computer Research Corp., 2000.

[Tho01] Thompson, H. et al.: XML Schema Part 1: Structures, May 2001 (available from http://www.w3.org/TR/xmlschema-1).

About the authors

Wook-Shin Han received the B.S. degree in Computer Engineering from Kyungpook National University in 1994, and the M.S. and Ph.D. degrees in Computer Science from Korea Advanced Institute of Science and Technology (KAIST), in 1996 and 2001, respectively. He is currently a full-time lecturer at Kyungpook National University. Previously, he was a postdoctoral fellow at KAIST. His research interests include object-oriented/object-relational databases, XML databases, and information retrieval. Email: wshan@knu.ac.kr.

Ki-Hoon Lee received the B.S. and M.S. degrees in Computer Science from Korea Advanced Institute of Science and Technology (KAIST), in 2000 and 2002, respectively. He is currently a Ph.D. candidate at KAIST. His research interests include object-oriented/object-relational databases, XML databases, and query optimization. Email: drlee@mozart.kaist.ac.kr.

Byung Suk Lee received the B.S. degree in Electronics Engineering from Seoul National University in 1980, the M.S. degree in Electrical Engineering from Korea Advanced Institute of Science and Technology in 1982, and the Ph.D. degree through Computer Systems Laboratory in Electrical Engineering from Stanford University in 1991. He is currently Assistant Professor of Computer Science at the University of Vermont. Previously he was Assistant Professor of Software Engineering at the University of St. Thomas, Supervisor at Datacom Global Communications, Member of Technical Staff at Bell Communications Research, and Senior Research Engineer at Gold Star Electrical Company. His research areas include databases, artificial intelligence, and information retrieval. Email: bslee@cs.uvm.edu.