A Meta-Model for the Storage of XML Schema using Model-Mapping Approach

Hoontae Kim\textsuperscript{1} \cdot Taesoo Lim\textsuperscript{2} \cdot Keunhee Hong\textsuperscript{3} \cdot Suk-Ho Kang\textsuperscript{4}

\textsuperscript{1}Dept. of Industrial and Systems Engineering, Daejin University, Pocheon, 487-711
\textsuperscript{2}LG Electronics PRC, Pyongtaek, 451-713
\textsuperscript{3}OpenTide Korea, Seoul, 135-845
\textsuperscript{4}Dept. of Industrial Engineering, Seoul National University, Seoul, 151-742

모델 매핑 접근법을 이용한 XML 스키마 저장 메타모델에 대한 연구

김홍태\textsuperscript{1} \cdot 임태수\textsuperscript{2} \cdot 홍근희\textsuperscript{3} \cdot 강석호\textsuperscript{4}

\textsuperscript{1}대진대학교 산업시스템공학과 / \textsuperscript{2}LG 생성기술원 / \textsuperscript{3}오픈타이드 코리아 / \textsuperscript{4}서울대학교 산업공학과

Since XML (eXtensible Markup Language) was highlighted as an information interchange format, there is an increasing demand for incorporating XML with databases. Most of the approaches are focused on RDB (Relational Databases) because of legacy systems. But these approaches depend on the database system. Countless researches are being focused on DTD (Document Type Definition). However XML Schema is more comprehensive and efficient in many perspectives.

We propose a meta-model for XML Schema that is independent of the database. There are three processes to build our meta-model: DOM (Document Object Model) tree analysis, object modeling and storing object into a fixed DB schema using model mapping approach. We propose four mapping rules for object modeling, which conform to the ODMG (Object Data Management Group) 3.0 standard. We expect that the model will be especially useful in building XML-based e-business applications.

\textit{Keyword:} XML Schema, meta-model, model mapping approach, DOM tree, object modeling

1. Introduction

Much business information has been generated on the web and companies have recognized a necessity for storing useful information created from business transactions. Since XML (eXtensible Markup Language) was highlighted as an information interchange format (IIF), much information has been transformed into XML documents. Then it was transferred between businesses and stored into a database. As a result, companies have been requesting the incorporation of XML into their databases (Bourret 2003). Those works are classified into a Relational mapping and an Object-Oriented mapping according to the type of database. RDB is preferred because it is

\textsuperscript{*}Corresponding author: Hoontae Kim, Dept. of Industrial and Systems Engineering, Daejin University, Sun 11-1, Sundandong, Pocheon, Gyeonggi, Korea, 487-711, Fax : 82-31-539-2000, E-mail: hoontae@daejin.ac.kr

Received August 2003, accepted July 2004 after 1 revision.
commonly used because a lot of advanced RDB techniques can be applied (Bertino and Catania, 2001; Florea and Kossmann, 1999; Shannuganadaram et al., 1999; Yoshikawa et al., 2001), whereas OODB is more proper for processing the complex data structure of XML (Christophides et al., 1994; Chung et al., 2001; Goldman et al., 1999; Lin et al., 2000). However, these previous approaches are limited to a specific database type. It is also the fact that most of those works are focused on XML instance and just some for DTD (Document Type Definition). However it is expected that XML Schema will substitute for DTD because of its richer resource sets for powerful information interchange as a document meta-data (Roy and Ramanujan, 2001).

This paper suggests a database-independent meta-model to store XML Schema document using DOM (Document Object Model) tree analysis, object modeling and model-mapping approach. Model-mapping approach uses a fixed DB schema so that it stores and easily updates meta-data of dynamic and structurally variant documents.

This paper explains related works and terminologies in chapter 2, suggests modeling procedures in chapter 3, explains each step of the process in chapter 4, 5, 6, and specific characteristics and future works are presented in chapter 7.

2. Related Works

2.1 XML Schema

XML meta-data describes the logical structure, contents and constraints of XML document. DTD derived from SGML (Standard Generalized Markup Language) is one of the XML meta-data description languages and has been widely used so far. However, it has a lot of problems such as insufficient data types, different syntax from XML, lack of namespace, etc. These problems have shown much limitation on the usage of DTD and led the birth of a new XML meta-data description language, the XML Schema.

XML Schema was approved as a W3C (World Wide Web Consortium) Recommendation in May 2001. It extends the document model of DTD and strengthens the functionalities as an IIF. It supports 19 built-in types including string, integer, float, double, Boolean, etc. and 25 derived built-in types. In addition, it effectively provides the flexibility of data type definition with user-defined type.

XML Schema is more comprehensive and efficient in several perspectives. As Roy and Ramanujan (2001) points out, its characteristics can be explained with the following 4 perspectives.

Data Type. XML Schema provides not only rich data types but also user-defined types. Therefore, it is easy to generate XML document having appropriate syntax and semantics.

Syntax. XML Schema follows XML syntax so that just one parser can process both XML meta-data and instances.

Reusability. XML Schema supports inheritance, that is, the partial or overall of existing XML Schema can be reused by inheritance.

Namespace. XML Schema prevents conflicts problems caused by naming duplication using namespace. In addition, it is easy to generate valid documents using various namespaces defined by several XML Schema.

2.2 Previous Researches

We classify previous researches into Object-Oriented models and Relational models according to the modeling method. In addition, there are structure-mapping and model-mapping approaches for database schema design of XML document. In the structure-mapping approach, a database schema (in this paper, meta-model means database schema) is defined for each XML document schema. It is suitable when we store a large number of XML documents that conform to a limited number of document structures and when the document structure is static. Contrastingly, in the model-mapping approach, database schemas represent constructs of the XML document model. That is, a fixed database schema is used to store the structure of all XML documents. It is appropriate for storing a large number of dynamic and structurally-variant XML documents like numerous sophisticated Web applications (Yoshikawa et al., 2001). Table 1 and Table 2 summarize some representative features and limitations of the model suggested by previous works.

This paper suggests an integrated model that is independent of the modeling method. As a solution, it applies the DOM tree analysis to extract structural characteristics from XML Schema and
performs object modeling to build database-independent data model, which is compliant with ODMG 3.0 Standard (Cattel and Barry, 2000). Furthermore, it designs a fixed database schema according to the model-mapping approach. Yhoshkawa et al. (2001) suggested modeling and querying methods using the model-mapping approach. They focused on DTD and tried to make a RDB model. However, this paper suggests a database-independent meta-model to store XML Schema document. In addition, while they enumerated all the paths and stored the informations in a table, we decomposed the link informations to store them more efficiently.

3. Modeling Method

3.1 Building Process

Our procedure to build XML Schema meta-

model is divided into three parts-first, DOM tree analysis of XML Schema and then, clustering of each node, second, four meta modeling rules applying to clustered nodes and the last, relational and object-oriented mapping for storing the generated ODMG 3.0 compliant schema into both databases according to the fixed database schema. <Figure 1> illustrates the overall procedure. In reality, meta modeling for primitive data type and facet is previously performed before the entire process, for it does not require a specific XML Schema document.

3.2 Fixed Database Schema

In model-mapping approach, a fixed schema is used to store the structure of all XML documents. Each XML document structure is stored as the data in the database according to the fixed database schema. We have designed two database schemas for both relational and object-oriented

<table>
<thead>
<tr>
<th>Model</th>
<th>Mapping methodology</th>
<th>Target document</th>
<th>Features</th>
<th>Limitations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Christophides et al., 1994</td>
<td>Structure mapping</td>
<td>DTD</td>
<td>- Create classes for all elements declared in DTD</td>
<td>- Abuse classes</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>- Exclude the storage of additional information.(comment, tag order, etc.)</td>
</tr>
<tr>
<td>Goldman et al., 1999</td>
<td>Structure mapping</td>
<td>XML instance</td>
<td>- Apply OEM(Object Exchange Model) data model</td>
<td>- Lack of document update</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- Introduce Dataguide about path information</td>
<td>- Need more space to store Dataguides than actual XML data</td>
</tr>
<tr>
<td>Chung et al., 2001</td>
<td>Structure mapping</td>
<td>DTD</td>
<td>- Apply inheritance: solve null value problem</td>
<td>- Exclude the storage of element order information</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Model</th>
<th>Mapping methodology</th>
<th>Target document</th>
<th>Features</th>
<th>Limitations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Florescu et al., 1999</td>
<td>Structure mapping</td>
<td>XML instance</td>
<td>- Use directed labeled graph.</td>
<td>- Expensive to re-compose data</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- Store node position and path information</td>
<td>- Expensive and complex to update</td>
</tr>
<tr>
<td>Shannugasun daram et al., 1999</td>
<td>Structure mapping</td>
<td>XML instance</td>
<td>- Classify XML document</td>
<td>- Exclude the storage of additional information. (comment, tag order, etc.)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- Data-centric</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- Document-centric</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- Hybrid representation</td>
<td></td>
</tr>
<tr>
<td>Bertino et al., 2001</td>
<td>Structure mapping</td>
<td>XML instance</td>
<td>- Store according to the structural characteristics of DTD</td>
<td>- Exclude the storage of element order information</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Yoshikawa et al., 2001</td>
<td>Model mapping</td>
<td>XML instance</td>
<td>- Use directed labeled graph</td>
<td>- Lack of full-text search</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- Store path information</td>
<td>- Lack of document update</td>
</tr>
</tbody>
</table>
databases, which are similar to each other. Table 3 shows specific tables and classes consisting of both fixed schemas.

![Diagram](image)

**Figure 1.** Building procedure of XML Schema meta-model.

4. DOM Tree Analyses

When XML Schema is loaded, DOM tree analysis is activated to grasp the document structure. Originally, DOM is the method to represent elements of XML hierarchically as a tree using several types of node (http://www.w3.org/TR/2003/WD-DOM-Level-3-Core-20030226). In this paper, we classify these nodes into 7 types including rootElement, subelement, comlextype, simpletype, attribute, primitive, and facet to analyze the structural characteristics concretely and precisely. Figure 2 illustrates an example of DOM tree analysis. General_info of Figure 2(a) is one of the complex types in RFQ.xsd. We composed a RFQ(Request For Quotation) document, which was introduced at the eCatalog technical committee of Korea Integrated Forum on Electronic Commerce on February 22, 2002, and Figure 2(b) is the result of DOM tree analysis for it.

![Diagram](image)

(a) General_info type

![Diagram](image)

(b) DOM tree analysis of General_info type

**Figure 2.** Example of DOM tree analysis.

<table>
<thead>
<tr>
<th>Tables in RDB</th>
<th>Classes in OODB</th>
<th>Role</th>
</tr>
</thead>
<tbody>
<tr>
<td>RootElementRepository</td>
<td>RootElement</td>
<td>Extent of root elements</td>
</tr>
<tr>
<td>ComplexTypeRepository</td>
<td>ComplexType</td>
<td>Extent of complex types</td>
</tr>
<tr>
<td>SubElementRepository</td>
<td>subElement</td>
<td>Extent of elements except a root element</td>
</tr>
<tr>
<td>AttributeRepository</td>
<td>Attribute</td>
<td>Extent of attributes</td>
</tr>
<tr>
<td>SimpleTypeRepository</td>
<td>SimpleType</td>
<td>Extent of simple types</td>
</tr>
<tr>
<td>PrimitiveTypeRepository</td>
<td>PrimitiveType</td>
<td>Extent of primitive types</td>
</tr>
</tbody>
</table>
This step analyzes the XML tree structure after loading the whole XML document. Even if the step requires large initial loading time, it is more efficient than other XML parsing approaches such as SAX (Simple API for XML) in the case of the high frequent access to XML document.

5. Meta Modeling

Meta modeling is the stage to generate data model independent of database. This paper suggests 4 mapping rules and these rules are compliant with ODMG 3.0 standard. First rule is regarding to class creation. The other rules specify type, element, built-in type, and attribute transformations.

Class Creation Rule. “Three classes are always generated. Those are RootElementRepository, SimpleType-Repository and ComplexTypeRepository classes.”

RootElementRepository, SimpleTypeRepository and ComplexTypeRepository classes store the information about root element, simple type and complex type, respectively.

Type and Element Transformation Rule. “ComplexType and SimpleType are defined as the attribute of ComplexTypeRepository and SimpleType-Repository class, respectively. Elements contained in the ComplexType are defined with struct declarer.”

A ComplexType is defined as an attribute of ComplexTypeRepository class after being re-defined with struct type declarer and the name and type name of the generated attribute have to be identical. <Figure 3> illustrates the transformation of type and element.

<Figure 3(a)> is some part of RFQ.xsd to show how element, complex type and simple type are transformed. As you see <Figure 3(b)>, root element RFQ, complex type and simple type are stored as an attribute of RootElementRepository, ComplexTypeRepository, SimpleTypeRepository, respectively. In addition, the type of attribute in the ComplexTypeRepository class is declared with struct type declarer.

Primitive Data Type Mapping rule. “A built-in primitive data type is transformed into an ODMG 3.0 literal. However, it is applied separately for common data type, constrained data type, decomposed data type and combined data type.”

Built-in data types are derived from anySimpleType that restricts anyType, the super type of XML Schema built-in data type hierarchy. It is classified into primitive data types derived directly from anySimpleType and data types extended from those primitive types. Refer to (http://www.w3.org/XML/Schema#dev) for the details.

As the XML Schema is not developed just for ODMG 3.0 standard, all data types of XML Schema don’t correspond to those of ODMG standard. The XML Schema supplies plentiful data types to express both document and data

---

![Figure 3. Type and element transformation.](attachment://figure3.png)
centric characteristics. On the other hand, the ODMG standard provides more generic types than the XML Schema. We divide the built-in data types into four categories as follows:

- **Common data type.** It is the data type that is directly mapped into the ODMG literal such as float, boolean, double, string, etc. It is possible to directly map a float type into a float literal. The `typedef` declarer specifies it as follows:

  ```
  typedef float float;
  ```

- **Constrained data type.** It is the data type that can be mapped by adding some restrictions to the ODMG literals such as nonPositiveInteger, positiveInteger, negativeInteger, nonNegative-Integer, etc. For instance, nonPositiveInteger is specified by long long type and restricted by the constraint that its value should be equal or less than zero. We design meta-model describing the type definition for the constrained data type and its indication for the stored procedure as follows:

  ```
  typedef long long nonPositiveInteger;
  class constraint_register{
    attribute list<constraint> constraints;
    void register();
  };
  struct constraint{
    string type;
    string check_method;
  };
  ```

- **Decomposed data type.** This type is mapped onto an ODMG 3.0 literal by combining several built-in data types of XML Schema. The XML Schema decomposes a date type into several types, such as gYearMonth, gYear, gMonthDay, gDay, and gMonth. On the contrary, ODMG 3.0 only provides a date literal. Therefore, the functions to combine those types need to be provided as follows:

  ```
  typedef date gYearMonth, gYear, gMonth,
  gMonthDay, gDay;
  class constraint_register {
    attribute list<constraint> constraints;
    void register();
    date date_combine(); // a method for combining several
    separated types.
  };
  ```

- **Combined data type.** It is the data type that is defined by re-combining other types such as NOTATION, NMTOKENS, IDREFS, ENTITIES and so on. It is defined by using struct, set and list of ODMG literal according to the combination type as follows:

  ```
  typedef set<QName> NOTATION;
  typedef list<NMTOKEN> NMTOKENS;
  typedef list<IDREF> IDREFS;
  typedef list<ENTITY> ENTITIES;
  ```

Attribute Transformation Rule. “Attribute is defined using struct type declarer.”

In XML Schema, ComplexType can have attributes and subelements. Those attributes need to be stored in the struct for the ComplexType because it has its own name and type. However, it is difficult to distinguish it from the elements in the complex type (refer to Type and element transformation rule). Therefore, it is defined with additional struct type declarer. **Figure 4**

![Figure 4](image)

**Figure 4.** Attribute transformation.
illustrates the example of attribute transformation. 
<Figure 4(a)> shows two attributes of RFQType complex type. They are declared with struct type declerar and then, defined as an attribute of Complex-TypeRepository like <Figure 4(b)>.

6. Database Mapping

After object modeling, the information extracted from XML Schema is stored according to the fixed database schema. Both databases store the information according to the following rules.

• RootElementRepository class is mapped into RootElementRepository table in RDB and RootElement class in OODB.
• ComplexTypeRepository class is mapped into ComplexTypeRepository table and ComplexType class.
• Complex type declared by struct is mapped into subElementRepository table and subElement class.
• Attribute type declared by struct is mapped into AttributeRepository table and Attribute class.
• SimpleTypeRepository class is mapped into SimpleTypeRepository table and SimpleType class.
• Primitive data type declared by typedef is mapped into PrimitiveTypeRepository table and PrimitiveType class.

<Figure 5> illustrates the tables, fields, primary keys, and foreign keys in the relational database schema as stated in 3.2, and the mapping example for <Figure 3(b)> meta model is shown in <Figure 6>. An OODB mapping is similar to RDB mapping, therefore we omit it.

![Figure 5. Relational database schema.](image)

7. Conclusions

This paper proposes a meta-model for the storing of XML Schema, which is independent of a specific database. To build our model, we used the model-mapping approach, DOM tree analysis and object-oriented modeling.

First, model-mapping approach fixes the meta-model so that it prevents from abusing classes and tables. It makes it easy to update dynamic XML document and as a result increases the efficiency of information management and retrieval.

![Figure 6. Storage of relational database.](image)
Secondly, a lot of time was spent on the DOM tree analysis. Although a lot of time was spent, we can precisely and concretely grasp the structural characteristics of the XML document. And last, object-oriented modeling easily catches the tree structure of XML document. We designed an effective and neutral meta-model following object-oriented ODMG 3.0 standard.

References


