02-MX

The 6th World Multiconference on Systemics, Cybernetics and Informatics

July 14-18, 2002
Orlando, Florida, USA

PROCEEDINGS

Volume VII

Information Systems Development II

Organized by IIIS
International
Institute of
Informatics and
Systemics

Member of
International Federation of
Systems Research IFSR

EDITED BY
Nagib Callaos
John Porter
Naphtali Rishe

Mapping from XML DTD to Semantic Schema*

Naphtali RISHE, Li YANG, Maxim CHEKMASOV,
Marina CHEKMASOVA, Scott GRAHAM, Alejandro ROQUE

High Performance Database Research Center
School of Computer Science
Florida International University
Miami, FL 33199, U.S.A.

ABSTRACT

The Extensible Markup Language (XML) is increasingly becoming a popular data exchange and representation format because of the need for enhancing data interoperability and exchangeability over the Web. Different approaches have been investigated on using database technology to store and access XML data. In this paper, we explain in detail a unique and complete mapping scheme to map a DTD to a Semantic Schema in Sem-ODM. The key idea is capturing the meta-schemata of both the DTD and Semantic Schema, and then mapping the basic constructs of DTD to their counterparts in the semantic schema, while preserving the structure and semantic information of the DTD. We were able to easily and naturally capture complex semantic information with the Semantic Schema, thus smoothing the mapping process.

Keywords: XML/DTD, Semantic Schema, Meta-schema, Structure Mapping, Semantic Mapping.

1. INTRODUCTION

As the World Wide Web is gradually becoming one of the most important communication media, the importance of improving the interoperability, easing the access to heterogeneous data sources and integrating data from different applications has been brought to light. As a result, the World-Wide-Web Consortium (W3C) [18] has defined a language called Extensible Markup Language (XML) [20]. XML is a subset of SGML (Standard Generalized Markup Language), which was originally designed for the purpose of large-scale electronic publishing and is now becoming a popular data exchange and representation format [18].

One of the issues that a lot of researchers and software vendors have been focusing on is XML storage. To date, three different repositories have been used. They are relational databases [9, 16, 7], object-oriented databases [2, 19] and semi-structured databases [8, 17]. Among this work, a great deal of effort has been put on using relational databases because of its mature technology. The general approach of using a relational database as an XML repository is first mapping a DTD/XML-Schema to a relational schema and then loading XML data into relational databases (RDBMS) for further querying. However, no matter what mapping techniques are used, it has proved to be very hard to avoid the common problem of fragmentation when using RDBMS. This is because relational data model is a two-dimensional table and column-based structure, whereas the XML data model is a graph-based structure. In order to fit the graph-based XML data into the table based relational model and keep the mapped relational database in a certain normalized form, fragmentation is inevitable.

In this paper, we describe an alternative approach for storing XML, which uses Semantic Binary Object-Oriented Database System (Sem-ODB) as the underlying XML repository. Sem-ODB was developed at the High-Performance Database Research Center (HPDRC) [14] and is based on a conceptual data model, the Semantic Binary Object-Oriented Data Model (Sem-ODM [13]). It has been successfully deployed for highly complex applications such as applications intended for storage and processing of large amounts of earth science observations and the TerraFly Geographic Information System (GIS) [3].

Sem-ODM has features of Object-Oriented data models (such as inheritance, oid, explicit description of relationships, and a high-level data model) while maintaining the simplicity of relational data models in the sense of using simple constructs. It is more natural to preserve structure as well as semantic information using Sem-ODM than the relational model, as illustrated in section 4. In addition, set-valued attributes and nested structures of XML can be easily represented using relations (like associations in OO model) in Sem-ODM, thus avoiding the common problem of fragmentation when using RDBMS to store XML. Furthermore, by using relation and user defined oid (surrogate) concepts in Sem-ODM, semantic information such as element and document order in XML is more suitable to representation in a semantic schema than in a relational schema.

In this paper, we describe in detail our approach of mapping DTDs to Sem-ODM semantic schemas. The basic idea is capturing the meta-schema of both DTD and the semantic schema, and then mapping the basic constructs of a DTD to their counterparts in a semantic schema, while preserving the structure and semantic information of the DTD. The mapping information is not hard-coded; rather it is generated dynamically during the mapping process and kept in a Sem-ODB for future query translation and reconstruction phases. The approach of capturing the meta-schemas of different data sources and storing the mapping information for future processing has been applied successfully in our SemWrapper project [12] and SemAccess project [15].

* This research was supported in part by NASA (under grants NAG5-9478, NAGW-4080, NAG5-5095, NAS5-97222, and NAG5-6830), NSF (CDA-9711582, IRI-9409661, HRD-9707076, and ANI-9876409), ONR (N00014-99-1-0952), and the FSGC.

The rest of the paper is organized as follows. Section 2 gives a brief overview of DTD, its basic constructs and its meta-schema represented in the form of Sem-ODM concepts. The meta-schema of a semantic schema is presented in Section 3. The structure mapping and semantic mapping are explained in Section 4 with examples. Section 5 presents some related work. Conclusions and future work are presented in section 6.

2. OVERVIEW OF DTD AND ITS META-SCHEMA

2.1 XML and DTD

XML is a subset of the Standard Generalized Mark-up Language (SGML). It is a tag language, but users have the freedom of customizing the tags. Thus it can be used to define other languages. An XML document is composed of a stream of text nested within pairs of matching open and close tags. The tags are called Elements. Each element may have attributes describing the element and contain sub-elements in its body. Each sub-element can have cardinality of only one, at most one (?), at least one (+), or many (*) to describe how many times that sub-element can appear within the body of its parent element. In addition, it can be specified whether the sub-elements should appear in some order (denoted by a "," between sub-elements) or not (denoted by "|"). In this way, an ordered, nested and hierarchical document can be formed.

The structure and constraints of XML documents are described by a Document Type Definition (DTD) [6]/XML Schema [21]. Figure 1 is a DTD example that was extracted from [16] and is used as a running example in this paper. The basic components of a DTD are elements and attributes. They are declared in the following form, respectively.

<!ELEMENT element_name element_content_type>
<!ATTLIST element_name attribute_name attribute_type default>.

```
<!ELEMENT book (booktitle, author)>
<!ELEMENT booktitle (#PCDATA)>
<!ELEMENT article (title, author*, contactauthor)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT contactauthor EMPTY>
<!ATTLIST contactauthor authorID IDREF #IMPLIED>
<!ELEMENT monograph (title, author, editor)>
<!ELEMENT editor (monograph*)>
<!ATTLIST editor name CDATA #REQUIRED>
<!ELEMENT author (name, addr)>
<!ATTLIST author id ID #REQUIRED>
<!ELEMENT name (first?, last)>
<!ELEMENT first (#PCDATA)>
<!ELEMENT last (#PCDATA)>
<!ELEMENT addr ANY>
```

Figure 1 DTD Running Example

DTD Simplification Assumption To ease the mapping process, we transform complex DTDs to simpler, but equivalent ones before performing the real mapping. In the subsequent sections, we will assume DTDs have been simplified by rules in [16, 5]. In the following section, we will review the components of DTD and present the meta-schema that we have devised to describe a DTD.

2.2 DTD Constructs

Note that notations of the Sem-ODM are used to illustrate DTD and Semantic Schema components throughout the rest of the paper (see section 3 for basic Sem-ODM concepts). Graphically, the rectangles represent *categories*. The dashed-arrow represents *ISA* links (inheritance/super-category subcategory relationships). The dashed-arrows point from *sub-category* to *super-category*. The *attributes* of a particular category are placed in the respective category rectangle with ranges placed after the ":" (semi-colon). The thick arrows (i.e. non-dashed) represent *relations* between categories. The cardinalities and constraints of relations are represented inside parentheses.

The basic constructs of a DTD are elements and attributes, which are depicted in Figure 2 in the form of a Sem-ODM Semantic Schema. Therefore, mapping can be considered from these two perspectives, element-related and attribute-related mapping.

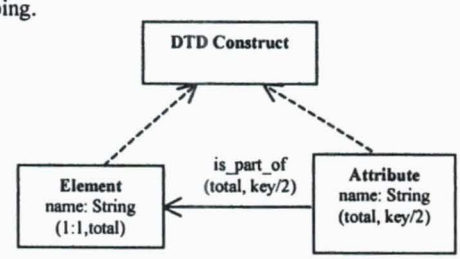


Figure 2 Sub-schema Representing DTD Construct

2.2.1 Meta-schema of Element According to the content type of elements, elements can be classified into EmptyContent, AnyContent, PCDataElement, Mixed, and Sequence, as illustrated in Figure 3. Recall that we simplify a DTD so that it does not contain choice content elements.

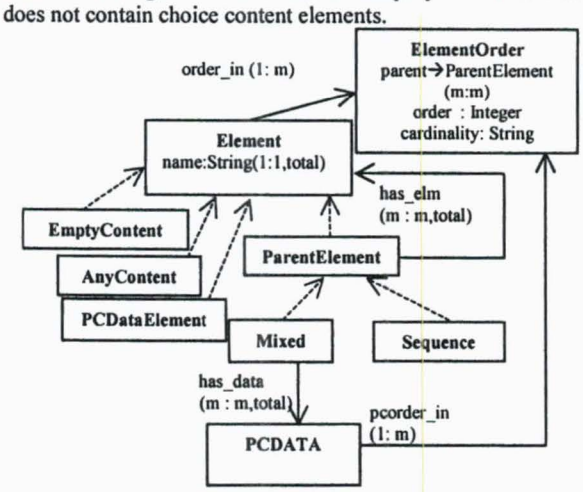


Figure 3 Sub-schema Representing DTD Element Meta-Schema

An EmptyContent element does not have any text between its start-tag and end-tag. An example of such an element is contactauthor in the running example. Elements of AnyContent may contain any content, which might be an element. For instance, element addr is declared as ANY content in Figure 1. Thus in the XML document, addr can have arbitrary XML fragments. In our classification, we use PCDataElement to differentiate elements declared as #PCDATA type and #PCDATA in a mixed content declaration. Elements first and last in our DTD are examples of PCDataElement. Category PCDATA in Figure 3 is used to represent the character data in a mixed content. Mixed content elements may contain character data as well as elements. For instance, the following example declares a mixed content element paragraph.

<! ELEMENT paragraph (#PCDATA* | figure*) >
<! ELEMENT figure (#PCDATA) >

By our definition, element figure will be regarded as PCDATAElement and #PCDATA within element paragraph is an object of PCDATA category. They will be treated differently in the following mapping process.

A Sequence element is an element which contains child-elements and an ordering among the sub-elements. For example, element book in the above DTD example consists of booktitle and author. Booktitle must appear in front of author in any XML document that conforms to this DTD. Because mixed content and sequence elements may have sub-elements, we call them ParentElements in our categorization.

Category ElementOrder in Figure 3 is used to capture the ordering and cardinality of an element in its parent elements. Attribute parent keeps the information about the parent element, thus has the range of ParentElement.

2.2.2 Meta-schema of Attribute An element may have a set of attributes associated with it. According to their types and defaults, attributes in DTDs can be further refined, as shown in Figure 4. An attribute can be of type CDATA, Tokenized, or Enumerate. Attributes can have REQUIRED, IMPLIED, DEFAULT, or FIXED declared as a default value.

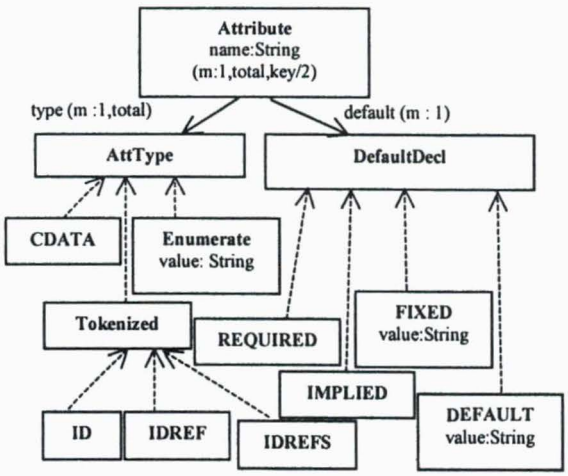


Figure 4. Sub-schema Representing DTD Attribute Meta-Schema

CDATA stands for character data, which is a string type. The REQUIRED option means that a value must be provided for the attribute for each element that this attribute belongs to. IMPLIED means no default value should be provided for this attribute. An attribute may have a default value in its declaration, which means if no value is provided for that attribute in the XML document, the default value will be assumed as its value. The FIXED option determines that the corresponding attribute can only have a fixed value when it appears in XML documents.

Two special attribute types worth mentioning are ID and IDREF(s). The former is used to uniquely identify each element. It is much like the key concept in the relational model, except that each element can only have one ID attribute and no multiple attributes are used to identify one element. An example is attribute id in element author. Each value of id can uniquely identify an author. An attribute of IDREF must have a value matching the value of some ID attribute. The attribute authorID of element contactauthor is of IDREF type.

3. OVERVIEW OF SEM-ODM CONSTRUCTS

There are two constructs, *category* and *relation*, used to describe a Sem-ODM. The categories and relations in Sem-ODM model tables and the foreign key relationship between tables in relational databases, respectively. Categories are like *Entities* in the Entity Relationship (ER) model, except that the *Attributes* in the ER model are represented as relations in Sem-ODM. A Category can either be a *Concrete Category* or an *Abstract Category*. Concrete Categories are categories like String, Number, and Boolean, etc. Abstract Categories are categories composed of abstract objects; an example of an sbstract category is person or book. There can be binary relations from an abstract category, which is called the *Domain* of the relation, to another category, which is called the *Range* of the relation, in a Semantic Schema. Relations from an abstract category to a concrete category in Sem-ODM are called attributes in ER model. Relations from an abstract category to an abstract category are just like associations in OO model. The constructs and relations between the constructs of a Sem-ODM can be illustrated in Figure 5.

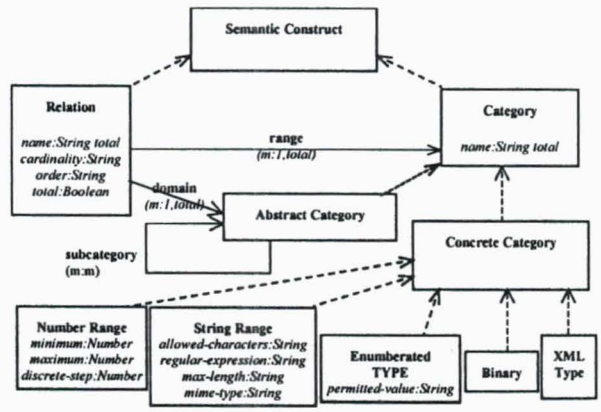


Figure 5. Subschema Representing Sem-ODM Meta-Schema

With the help of relations in Sem-ODM, the nested structure of XML can be represented in a Semantic Schema. Sem-ODM also has the concept of *Cardinality* of relations, which can be used to capture the same concept in DTDs. There is no ordering concept in Sem-ODM. We would have to extend the relations in Sem-ODM with ordering to support this feature of XML. This is illustrated in the Relation category in the above figure in which an order attribute is added. In addition, an XMLType category is added as a Concrete Category in the above figure. This category is used to represent ANYContent in DTDs.

4. STRUCTURE AND SEMANTIC MAPPING

In order to store XML in Sem-ODB, we need to map the constructs of a DTD to their counterparts in a Semantic Schema. During the mapping process, we preserve the structure and semantic information defined in the DTD for future query translation and result construction. By doing structure mapping, the elements and attributes are mapped to categories and relations in Sem-ODB. Performing the semantic mapping ensures that the mapping process is complete, i.e., all the semantic information such as cardinality, nullable, and reference is set in the Sem-ODB.

4.1 Structure Mapping

Three kinds of structure mapping are performed: element, relationship, and attribute. To make the algorithm easier to understand, we introduce the following naming conventions:

(1) Each element E in the DTD is represented by its name (e.g. element "book" is represented by book).
(2) The category, which corresponds to element E in the DTD is represented as E in the Semantic Schema$_c$ (e.g. the category that's mapped from element "book" is represented as book$_c$).
(3) Relations which are mapped from some attributes or elements E in the DTD are represented as E_R in the corresponding Semantic Schema.

In addition, for simplicity, we use the following notation to represent a relation r with domain category D, range R, cardinality c and totality t: r:D→R (c,t).

4.1.1 Element Mapping An element of the DTD will be mapped to a category or inlined as an attribute of a category according to its content type and whether or not it is a child element. The following is the detailed description.

(1) For every Element E of Sequence, EmptyContent, and Mixed type is mapped to a Category E_C in the Semantic Schema. For example, book in Figure 1, which is a sequence element, is mapped to a category called book$_c$ in the semantic schema.
(2) For every Element E of AnyContent type,
   a. If E has only one parent element P and the cardinality of E in P is not * or +, then E is mapped to a relation E_R from the category E_P, which is the category corresponding to the element P in the Semantic Schema, to XMLType, i.e., E_R: E_P →XMLType.
   For example, element addr is mapped to addr: author→XMLType.
   b. Otherwise, E is mapped to a Category E_C with a relation R from E_C to XMLType, i.e., R: E_C→XMLType
   For example, element test declared in <!ELEMENT test ANY> will be mapped to a category called test with relation data: test→XMLType.
(3) Similar to (2), for every Element E of PCDATAElement type,
   a. If E has only one parent element P and the cardinality of E in P is not * or +, then E is mapped to a relation E_R from the category E_P, which is the category corresponding to the element P in the Semantic Schema, to String, i.e., E_R: E_P →String.
   For example, element first is mapped to an attribute of category name (first: name→String).
   b. Otherwise, E is mapped to Category E_C with a relation R from E_C to String, i.e., R: E_C→String.
   For example, element title is shared by element article and monograph; therefore, it is mapped to a category called title with relation data: title→String.

   The idea behind (2) and (3) is if E is shared by many elements, or it may appear in its only parent more than once, it should be mapped to a category and let all the parents set up a relation with it. Otherwise, let it be an attribute of its parent category.
(4) For each PCDATA, map it as a relation from the category corresponding to the mixed element to String. For example, the #PCDATA in <! ELEMENT paragraph (#PCDATA| figure) *> will be mapped to data: paragraph→String(m:m).

4.1.2 Relationship Mapping Some important structure information in a DTD lies in the implicit relationships between elements and their sub-elements and between elements and their attributes. Unlike DTD to relational schema mapping, in which this kind of information is implicitly expressed by the key and foreign key relationship or table-column relationship, a semantic schema explicitly represents them as a relation.

The mapping is described in the following.

(1) If any of the above rules can be applied, then skip the following. For example, booktitle is a sub-element of element book. Since it's a PCDATAElement, it is mapped as an attribute of category book according to rule (3) in section 4.1.1. The following two rules are not applicable.
(2) Every sub-element E_i of element E of Sequence type is mapped into a relation from E_c to E_{ic}, where E_c is the category corresponding to E and E_{ic} is the category corresponding to E_i. For example, the relationship between elements book and author is mapped as a relation called book_author (see Figure 6)
(3) Similarly, every sub-element E_i of element E of Mixed type is mapped into a relation from E_c to E_{ic}, where E_c is the category corresponding to E and E_{ic} is the category corresponding to E_i.

Note that in the above sections 4.1.1 and 4.1.2, if a relation is created in the Semantic Schema, its cardinality and totality have to be decided according to the rules in section 4.2.1.

4.1.3 Attribute Mapping Attributes in DTDs are mapped to relations, whose domain is the category corresponding to the element that this attribute belongs to and whose range is a Concrete Category. The mapping rules are generalized as follows.

(1) For <! ATTLIST E att CDATA>, where E is an element and att is its attribute of type CDATA. Attribute att is mapped to a relation att$_R$: E_C → String(m:1) (e.g. attribute name of element editor, see Figure 6).
(2) For <! ATTLIST E att eval1 | eval2 | eval3 >, where E is an element and att is its attribute of type Enumerate. Attribute att is mapped to a relation att: E_C → Enumerate(m:1).
(3) For <! ATTLIST E att ID>, where E is an element and att is its attribute of type ID. Attribute att is mapped to a user defined oid of category E_C when we don't consider the document order of XML. However, if we want to consider that order, we have to map it to a relation att: E_C → String (m:1). (e.g. attribute id in element author, see Figure 6)
(4) For <! ATTLIST E att IDREF>, where E is an element and att is its attribute of type IDREF. Attribute att is mapped to a relation att: E_C → E_{id} (m:1), where E_{id} is coresponding to the element that att is pointing to in the DTD. For example, attribute authorID of element contactauthor is of IDREF type and is mapped to a relation between contactauthor and author, i.e., authorID: contactauthor → author (m:1). Note that since IDREF(s) is untyped, the program could not tell which element that this IDREF attribute is pointing to. We have to explicitly set up the above relation.
(5) For <! ATTLIST E att IDREFS>, where E is an element and att is its attribute of type IDREFS. Attribute att is mapped to a relation att: E_C → E_{id} (m:m), where Eid is coresponding to the Element that att is pointing to in the DTD.

Furthermore, if the default type of the attribute is #REQUIRED, the totality of the relation is total.

4.2 Semantic Mapping

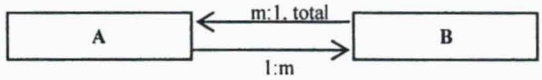
4.2.1 Cardinality Constraints The cardinality of a sub-element in a DTD represents the occurrence of the sub-element within its parent element. To our knowledge, most known DTD-relational mapping algorithms either did not explicitly explain the mapping of cardinality or did so rather complicatedly as in [10]. It is rather easy in DTD to Sem-ODM mapping, since there is the cardinality concept, such as m:1, 1:m, m:m, 1:1, defined in the Sem-ODM. For only one and at least one semantics in a DTD, we use total concept to represent the semantics.

The mapping of sub-elements cardinality is as follows:

(1) Cardinality "only one" is mapped as a relation with m:1, total (e.g. element last)

(2) Cardinality ? is mapped to m:1 (e.g. element first in Figure 1 has ? cardinality within its parent element name, thus it is mapped to first: name→String (m:1))

(3) Cardinality * is mapped to m:m except when there is a cycle between two elements that have the parent and child relationship and the cardinality in one of them is only one or at most one. In the latter case it is mapped to 1:m. For example, suppose we have <!ELEMENT A (B*) > and <!ELEMENT B(A) > in a DTD. In this situation, we will have



The reason that we map * to 1:m in this case is as follows. When only one or at most one semantics appears, we would derive a conflict if we still mapped * to m:m. For instance, suppose we map the above DTD to A→B (m:m) and B→A (m:1, total), then by B→A (m:1) , we derive B→A is 1:1, however A→B (m:m) tells us that it is m:1 instead. An example shows this mapping in Figure 1 is between element editor and monograph (see Figure 6)

(4) Cadinality + is mapped to m: m (total)

4.2.2 Inclusion constraints The IDREF and IDREFS in DTDs actually represent the foreign key relationship, which is a kind of inclusion constraint, in relational database systems. In Sem-ODB, we do not have this concept, but we can map this constraint to a relation that points to the category which has the ID attribute, as we explained earlier in section 4.1.3. This mapping is more accurate than key and foreign key mapping as seen in the relational model, because a key could be a composite key. However, in XML ID by itself uniquely defines an element. For example, the id of element author is of ID type, it is mapped to a uid of author, if we do not consider document order. The authorID in element contactauthor is of IDREF type. It is mapped to a relation which points to author.

4.2.3 NULL and not NULL As discussed previously, Attributes in DTDs may have default options. We map the #REQUIRED and #IMPLIED concept to total or not total and #FIXED and #DEFAULT to default value in Sem-ODM. The mapping rules about the default options are generalized as follows:

(1) Attribute *A* with #REQUIRED default option: the corresponding relation is total (i.e., not null)

(2) Attribute *A* with #IMPLIED default option: the corresponding relation is normal (i.e., nullable)

(3) Attribute *A* with #FIXED default option: set the fixed value as default for this relation in Sem-ODB

(4) Attribute *A* with #DEFAULT default option: set default value for this relation in Sem-ODB

4.2.4 Document Order and Element Order One important feature of XML is that it may be viewed as an ordered document, i.e., a *document order* exists among the elements of an XML document. In addition, there's an *element order* concept in XML, which is captured in the DTD/XML-Schema. XML documents can root at any element as long as the relationship between elements and sub-elements is kept valid. To our knowledge, most of the papers on XML mapping schemes either did not consider the ordering issue or simply mentioned that it is easy to enhance their scheme with ordering concept. However, in most DTD→relational schema mapping approaches, in order to keep such information, at least two more columns need to be created in the relational tables to represent the parent-child relationship (element order) and ordering among tuples (document order) in each table representing an element, as in [11, 16]. In Sem-ODB, these concepts are much easier to capture. The element order is first kept in the ElementOrder Category in the meta-schema in Figure 3, and is later mapped as an attribute of each relation in Sem-ODM. As for the document order, since in Sem-ODB we can provide a uid to each abstract or binary object, it can be represented by the ordering of such uids, as long as we relax the mapping of attributes of ID type as mentioned in Section 4.1.3.

Based on the above mapping rules, we can transform the DTD in Figure 1 into the Semantic Schema shown in Figure 6. Note that in Figure 6, the relations are enhanced with ordering which is denoted by an ordering number along with the relation name. And the default cardinality in Sem-ODM is m:1.

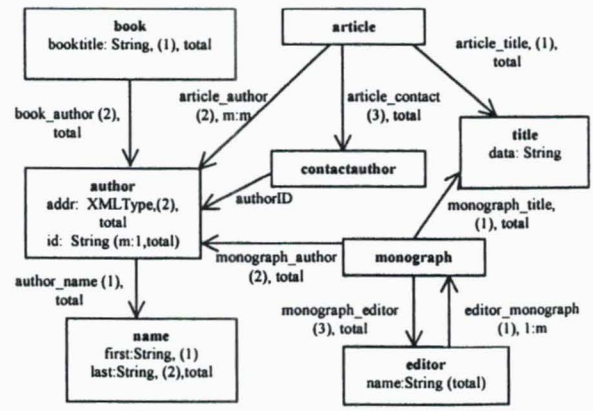


Figure 6. Semantic Schema Representation of the DTD Example

Through the transformation, the structure and semantic information of a DTD are more vividly represented in the Semantic Schema. Nested structures are split and distributed into categories and relations. Relationships between elements and sub-elements are expressed explicitly. ID and IDREF concepts are seamlessly integrated into Sem-ODM. Semantic information such as cardinality and default value are mapped to corresponding concepts in Sem-ODM. We were able to easily capture complex semantic information with the semantic schema.

5. RELATED WORK

Many XML to relational model mapping schemes have been proposed in recent research projects [7, 16, 9, 10, 11] or applied in commercial products [4, 1]. [7] viewed XML as an ordered and labeled graph and derived relational tables according to the edges between nodes in the XML graph. This approach does not need a pre-existing DTD to help with the generation of relational tables. However, because we believe it is beneficial to have a DTD to describe XML documents, and many applications which use XML document as exchange format require the presence of a DTD, our approach assumed the existence of DTDs for XML documents as did the approaches used in [16, 10]. However, [16, 10] generated relational tables according to the description of the DTD and the normalization requirement of the relational model. Our approach first captured the meta-schema of a DTD and a semantic schema, which is similar to the approaches used in [11, 9], and then performed the actual mapping and stored the mapping information in a Sem-ODB. [11] proposed a generic relational schema to store the structure information about any XML document so that relational data sources can be described as views over the generic schema. X-RAY in [9] introduced meta schemas for both DTD and relational schemas, defined the mapping between them, and stored the mapping information for resolving schema heterogeneity. Most of work, except [10], only explained the mapping from a structural point of view, not the semantics of the mapping.

Similar to that taken in [10], our approach described the semantic mapping as well as structural mapping in detail. However, since we are using a different and more powerful data model, Sem-ODM, the semantics can be more vividly and naturally represented in our approach. Our mapping is easier compared to the one in [10], since some semantic concepts of DTD can be found in Sem-ODM. We also provided a complete solution for handling document order and element order. In addition, by capturing the meta-schemas of both DTD and Semantic Schema and storing them as well as mapping information in Sem-ODB, the autonomy of both data sources is preserved.

6. CONCLUSIONS AND FUTURE WORK

In this paper, we explained in detail a unique and complete mapping scheme to map a DTD to a Semantic Schema in Sem-ODM. The mapping scheme is complete in the sense that both structure and semantic information is mapped. The mapping is natural because Sem-ODM has very similar concepts to XML.

As an extension to the current work, we plan to implement the above mapping scheme to further evaluate its efficiency in generating the meta-schema, storing mapping information, and loading XML document into Sem-ODB. In addition, we are interested in translating XML queries into SQL and evaluating the mapping through query translation and optimization.

7. REFERENCES:

[1] Sandeepan Banerjee, Vishu Krishnamurthy, Muralidhar Krishnaprasad, Ravi Murthy, "Oracle*8i* – The XML *Enabled* Data Management System", ICDE 2000.

[2] V. Christophides, S. Abiteboul, S. Cluet, M. Scholl, "From Structured Documents to Novel Query Facilities", Proceedings of the 1994 ACM SIGMOD International Conference on Management of Data, Minneapolis, Minnesota, May, 1994.

[3] Naphtali Rishe, " TERRAFLY: A High-Performance Web-based Digital Library System for Spatial Data Access", ICDE Demo Session, 2001

[4] Josephine Cheng, Jane Xu, "XML and DB2", ICDE 2000.

[5] A. Deutsch, M. Fern'ndez, and D. Suciu, "Storing Semistructured Data with STORED", Proceedings of the ACM SIGMOD International Conference on Management of Data, Philadelphia, Pennsylvania, USA, June 1999.

[6] J. Bosak, T. Bray, et. al., "W3C XML Specification DTD",http://www.w3.org/XML/1998/06/xmlspec-report.html .

[7] D. Florescu, D. Kossmann, "Storing and Querying XML Data Using an RDBMS", IEEE Data Engi-neering Bulletin, Special Issue on XML, Vol. 22, No. 3, September, 1999.

[8] R. Goldman, J. McHugh, and J. Widom, "From Semistructured Data to XML: Migrating the Lore Data Model and Query Language", Proceedings of the 2nd International Workshop on the Web and Databases (WebDB '99), Philadelphia, Pennsylvania, June 1999.

[9] Gerti Kappel, Elisabeth Kapsammer, S. Rausch-Schott, Werner Retschitzegger, "X-Ray -Towards Integrating XML and Relational Database Systems", 19th International Conference on Conceptual Modeling, Salt Lake City, Utah, USA, October, 2000.

[10] Dongwon Lee, Wesley W. Chu, "Constraints-preseving Transformation from XML Document Type Definition to Relational Schema", Proc. 19th Int'l Conf. on Conceptual Modeling (ER), Salt Lake City, Utah, October, 2000.

[11] Ioana Manolescu, Daniela Florescu, and Donald Kossmann, "Pushing XML queries inside relational databases", Tech. Report No. 4112, INRIA.

[12] Napthali Rishe, Jun Yuan, Rukshan Athauda, et al, "SemWrap: A semantic wrapper over relational databases, with substantial size reduction of user's SQL queries", International Conference on Extending Database Technology (EDBT), Konstanz ,Germany, March 27-31,2000.

[13] Naphtali Rishe, "Database Design: The Semantic Modeling Approach", McGraw-Hill, 1992.

[14] Rishe N., Sun W., Barton D., Deng Y., Orji C., Alexopoulos M., Loureiro L., Ordonez C., Sanchez M., Shaposhnikov A., "Florida International University High Performance Database Research Center". In *SIGMOD Record*, 24 (1995), 3, pp. 71-76.

[15] Naphtali Rishe, Jun Yuan, Rukshan Athauda, et al., "SemanticAccess: Semantic Interface for Querying Databases", Proceeding of the VLDB conf. , Cairo, Egypt, 2000.

[16]Jayavel Shanmugasundaram, et al, "Relational Databases for Querying XML Documents: Limitations and Opportunities", Proceedings of the 25th Int. Conf. On Very Large Data Bases (VLDB), Edinburgh, Scotland, UK. 1999.

[17] Tamino XML Database Home Page, http://www.softwareag.com/tamino/.

[18] The World Wide Web Consortium, http://www.w3.org.

[19] Excelon Home Page, http://www.exceloncorp.com/.

[20] "Extensible Markup Language (XML) 1.0 (Second Edition)", W3C Recommendation, October 2000, http://www.w3.org/TR/2000/REC-xml-20001006, work in progress.

[21] "XML Schema Part 0: Primer", W3C Recommendation, May 2 2001, http://www.w3.org/TR/xmlschema-0/.