Transforming Sem-ODM Semantic Schemas to DTDs

Li Yang
Department of Computer Science
University of West Georgia
Carrollton, GA 30118, USA
1-678-839-6656
lyang@westga.edu

Naphtali Rishe
High Performance Database Research Center
School of Computer Science
Florida International University
Miami, FL 33199, USA
1-305-348-2025
rishen@cs.fiu.edu

ABSTRACT

Exporting data in traditional databases as XML documents so that non-XML data can be accessed and exchanged seamlessly among applications has been studied in a number of projects. In this paper, we present an algorithm that automatically derives a Document Type Definition (DTD) from a Semantic Schema of the Semantic Binary Object-Oriented Data Model (Sem-ODM) based on the formal definitions of DTDs and Semantic Schemas. The structure and semantic information of a Semantic Schema is captured and expressed naturally in a DTD without applying any complicated operators or scanning the actual data stored in the database. Furthermore, the formalization of the two schemas connects the two data models and helps us better understand the capabilities brought about by publishing Sem-ODB data as XML.

Categories and Subject Descriptors

H.2.1 [**Logical Design**]: Data models, Schema and subschema; H.2.5 [**Heterogeneous Databases**]: Data translation

General Terms

Algorithms

Keywords

DTD, Semantic Schema, Formal Definition, Schema Transformation

1. INTRODUCTION

While XML (eXtensible Markup Language) [1] becomes the *de facto* standard for data representation and exchange on the World Wide Web (WWW), most of the data in the world still resides in traditional databases and legacy systems. These systems are backbones of lot of businesses, thus it is not realistic to give them up completely. Yet, it is feasible to provide XML interfaces to those systems. In this way, one can access non-XML applications using available XML tools. And non-XML data in the systems can be published as XML so that different applications can exchange their data seamlessly with each other. In this paper, we study the conversion in this respect. In particular, we study the transformation of a Semantic Schema of the Semantic Binary Object-Oriented Data Model (Sem-ODM) [10] to a DTD (Document Type Definition) [1] in order to facilitate publishing data in Semantic Binary Object-Oriented Database System (Sem-ODB) as XML.

Sem-ODB was developed at the High-Performance Database Research Center (HPDRC) at Florida International University and is based on a conceptual data model, Sem-ODM. As a fully functional multi-user object-oriented DBMS, Sem-ODB 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) [11].

In this paper, we present an algorithm that automatically derives a DTD from a Semantic Schema based on the formal definitions of DTDs and Semantic Schemas. Because Sem-ODM is a high-level and conceptual level data model, which supports inheritance, explicit relationships and 1:m attributes, and other features, it is simpler and easier converting a Semantic Schema to a DTD than converting a relational schema to a DTD. The structure and semantic information of a Semantic Schema is captured and expressed naturally in a DTD without applying any complicated operators or scanning the actual data stored in the database to come up with the nested structure in the resulting DTD as NeT did in [4, 5]. Furthermore, the formalization of the two schemas connects the two data models and helps us better understand the capabilities brought about by publishing Sem-ODB data as XML.

The rest of the paper is organized as follows. We first present related work in Section 2. We then introduce the formal definitions of DTDs and Semantic Schemas in section 3. Section 4 formally describes the mapping from a Semantic Schema to a DTD. Section 5 concludes this paper and points out the future work.

2. Related Work

Publishing data in traditional databases into XML has been studied in various projects. Most DBMS commercial products, such as MS SQL Server 2000, IBM DB2, among others, provide support for publishing relational data as XML. However, they all either generate a simple flat mapping where the resulting XML doesn't have reasonable nested structure or require users explicitly specify the mapping via a means such as an annotated language or macro file. For instance, DB2 databases can export their data as XML via IBM Net.Data Macro, which is basically a macro file with programmer-specified SQL query and output XML structure. SQL Server [12] provides XML virtual views over relational tables via an annotated XDR (XML-Data Reduced) schema language, in which annotations are used to specify the mapping between relational tables/columns and XML elements/attributes. Our approach is automatic and doesn't require users to specify the mapping scheme, and it takes into account the hierarchical structure of DTD when mapping Sem-ODM schemas.

Lee et al. [4, 5] devised two algorithms, NeT and CoT to map relational schemas to DTDs. To avoid a flat DTD, Net uses the nest operator and can only deal with one table. Cot, on the other hand, is able to convert multiple interconnected relational tables into DTDs based on inclusion dependencies obtained from the underlying relational databases. Like Cot, we rely on inclusion dependencies to generate the hierarchical structure of XML. However, since inclusion dependencies are explicitly expressed as relations between categories in the Sem-ODM, our conversion can be performed in a much simpler way than theirs. Additionally, NeT is only applicable in a single table case and is costly since it requires to scan the entire data to determine whether or not a column in a table should be nested and have multiple occurrence * or +. Though CoT deals with a complete relational schema case, to generate a nested structure for columns that are not involved in inclusion dependencies, NeT has to be performed first. Our algorithm can handle both single categories as well as a database schema, which involves the interleaving of multiple categories (similar to tables in the relational model). This is enabled by Sem-ODM's support for 1:m attributes. These attributes can be naturally translated into sub-elements with * or + occurrence.

Several XML schema languages, such as DTD and XML Schema [3] among others, have been proposed to describe the structure and semantics of XML documents. [8] formally described several XML schema languages (DTD, XML Schema, RELAX [9], and others) based on regular tree languages. It represented a DTD as a local regular tree grammar, whereas [14] represented a DTD as an extended context free grammar and [2] presented a DTD in terms of Description Logic. Lee et al. [4] formalized relational schemas and DTDs, and presented a nesting-based translation algorithm to transform a relational schema into a DTD. In [6], Mani et al. formally defined XGrammar, which combines the features of several XML schema languages, and studied its data modeling capability and performed the transformation between XGrammar and an extended ER model. Our definition is similar to the one in [4] and has been influenced by the formalism presented in [6]. However, [6] formalized not just one particular XML schema language, but rather a core set of features for several XML schema languages; our work is more specific. Moreover, we are concerned with transformation from a Sem-ODM Semantic Schema to a DTD, not from a relational schema to a DTD.

Another closely related research direction is storing XML data in traditional databases, which involves the opposite direction of the conversion. We have finished the research on DTD to Semantic Schema mapping. Our study shows that storing XML in Sem-ODB is feasible and efficient in terms of shorter and fewer join queries in translating XML Query into Semantic queries compared to traditional approaches. Since it is outside the scope of this paper, we do not discuss further.

3. Formal Definitions of DTDs and Semantic Schemas

3.1 Definition of DTDs

Since the appearance of DTDs, many XML schema languages such as XML Schema, and RELAX, among others, have been proposed to describe the structure and semantic constraints of XML documents. Our focus here is on DTDs due to their simplicity and wide acceptance. Our study has been influenced by the research in [6, 2, 4]. For simplicity, we do not consider ENTITY, ENTITIES, NMTOKEN, and NMTOKENS attribute types in this paper.

Before proceeding with the definition of DTDs, we first present some notation assumptions. Assume $\hat{A}$ is a finite set of attribute names, $\hat{E}$ is a finite set of element names, $\hat{\tau}$ is a finite set of attribute types permitted in a DTD and $\hat{\tau}$ ::= {CDATA,ENUM, ID, IDREF, IDREFS}, $\hat{d}$ is a set of default types that are allowed in a DTD attribute and $\hat{d}$ ::={IMPLIED, REQUIRED, FIXED} or ϵ which represents the case where no default type is specified, and that $\hat{u}$ is a set of default values of attributes where $\hat{u}$ ::= {u | u is a string or an integer allowed in a DTD, or ϵ }. Note that u =ϵ represents no default value is provided.

(Definition 1) A *Document Type Definition* (DTD) is formally denoted by a 4-tuple $\mathcal{G} = (E, A, S, P)$, where:

- E is a finite set of element names, representing elements, $E \subseteq \hat{E}$;
- A is a finite set of attributes. Each item of A is of the form $X(a{:}\ \tau{:}\ d{:}\ v)$, where $X \in E$, $a \in \hat{A}$, $\tau \in \hat{\tau}$, $d \in \hat{d}$, $v \in \hat{u}$, representing a is an attribute of element X with τ as the attribute type, d as the default type, and v as the default value of a;
- S is a finite set of start symbols, i.e., a set of root elements;
- P is a set of element definition rules in the form of $X \longrightarrow r$, where $X, Y \in E$ and r is the content model of X and can be generalized in the following abstract syntax:

$$r ::= \epsilon \mid Y \mid PCDATA \mid (\ r\ ) \mid r|r \mid r,r \mid r? \mid r* \mid r+$$

In the above definition, ϵ represents the empty string (i.e. EMPTY content), PCDATA represents content that consists of any string, ',' represents concatenation (Sequence content), '|' represents Choice content, '?' represents zero or one occurrence of r, '*' represents zero or more occurrences of r, and '+' represents one or more occurrences of r. Another content model, 'ANY', is not specified in the above syntax. Elements of ANY content can contain any information, tagged or untagged, i.e., it can be denoted as X*, where $X \in E$ and X can be of any content defined above.

For example, the DTD in Figure 1 which is extracted from [13] and slightly modified can be represented formally as $\mathcal{G}_1 = (E, A, S, P)$, where:

- E={*publication, book, article, title, author, contactauthor, name, first, last, address*}
- A={*contactauthor(authorID:IDREF:IMPLIED:*ϵ*), author(id:ID:REQUIRED:* ϵ*)*}
- S = {*publication*}

- *P*= {*publication* $\rightarrow$ *(book*, article*), book* $\rightarrow$ *(title,author), title* $\rightarrow$ *PCDATA, author* $\rightarrow$ *(name, address), name* $\rightarrow$ *(first?, last), first* $\rightarrow$ *PCDATA, last* $\rightarrow$ *PCDATA, address* $\rightarrow$ *ANY, article* $\rightarrow$ *(title, author*, contactauthor), contactauthor* $\rightarrow$ *(ϵ)* }

```
<!DOCTYPE  publication  [
            <!ELEMENT publication (book*, article*)>
            <!ELEMENT book (title, author)>
            <!ELEMENT title  (#PCDATA)>
            <!ELEMENT author (name, address)>
            <!ATTLIST author   id ID #REQUIRED>
            <!ELEMENT name (first?, last)>
            <!ELEMENT first (#PCDATA)>
            <!ELEMENT last  (#PCDATA)>
            <!ELEMENT addressANY>
             <!ELEMENT article (title, author*, contactauthor)>
            <!ELEMENT contactauthor EMPTY>
            <!ATTLIST contactauthor authorID IDREF   #IMPLIED>
]>
```

Figure 1. DTD Running Example

3.2 Definition of Semantic Schemas

The Sem-ODM (Semantic Binary Object-Oriented Data Model) is a high-level data model. As a conceptual level data model, it can mirror the real world enterprise scenarios naturally as the ER (Entity Relationship) model does. It supports 1:m attributes, which makes natural nesting an attribute as the sub-element with * or + occurrence of its parent element. In addition, it has some advantages of the Object-Oriented data model, such as inheritance, oids, and explicit relationships among objects, etc.

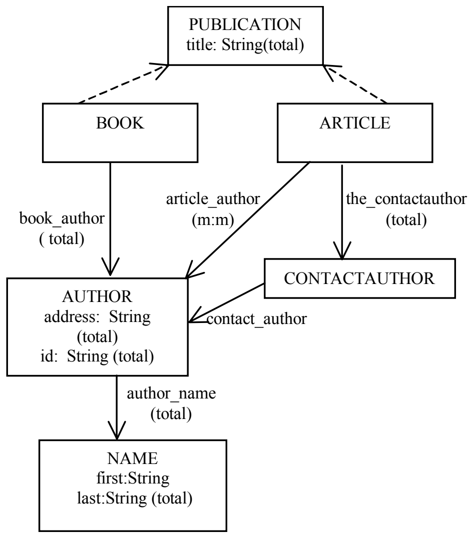

Figure 2. Semantic Schema Example Representing Publication

The basic constructs in the Sem-ODM are *Categories* and *Relations*, which are like Entities and Relationships in ER model, respectively. There are two kinds of categories in the Sem-ODM, *Concrete Categories* and *Abstract Categories*. Concrete Categories are atomic data types such as String, Number, and Boolean, among others. Abstract Categories are categories composed of abstract objects, such as person and book. The relations in a Semantic Schema are binary. Each of them is created from an abstract category, which is called the *Domain* of the relation, to another category, which is called the *Range* of the relation. Relations from an abstract category to a concrete category are called attributes in the ER model (we also call them *attributes* in a Semantic Schema). Relations from an abstract category to an abstract category are just like associations in an Object-Oriented model. Graphically, in the Sem-ODM, categories are represented by rectangles. Solid arrows, starting from the domain categories and ending at the range categories, are used to represent non-attribute relations. Inheritance is represented by dashed arrows from sub-categories to super-categories. Attributes are represented inside category rectangles with a colon (:) delimiting the attribute's name and type. Cardinality and other constraints (such as totality[1]) of a relation are placed alongside its type in parentheses. Figure 2 shows an example Semantic Schema for publications. For example, *publication* is a super-category, which has two sub-categories: *book* and *article*. *publication* has a total attribute called *title* with a range of Concrete Category *String*. The category *book* has a relation called *book_author* pointing to the category *author*. Note that in a Semantic Schema, relations without specifying cardinalities have *m:1* cardinality by default.

We now formally define the Sem-ODM model. Before we start, we assume that $\widehat{C}_a$ is a finite set of abstract category names, $\widehat{C}_c$ is a finite set of concrete category names, $\widehat{R}$, is a finite set of relation names, and $\widehat{V}$ is a finite set of strings representing the values of cardinality and totality, and $\widehat{V}$ ::= {m_1, m_m, 1_m,1_1, total, not_total}.

(Definition 2) A *Sem-ODM Semantic Schema* can be formally denoted as a 4-tuple $\mathcal{H}=(C_a, W, C_c, R)$ where:

- C_a is a finite set of abstract category names, $C_a \sqsubseteq \widehat{C}_a$;
- W is a finite set of inheritance relationships and each item in W has the form of $(O, S_1, S_2..S_n)$, where $O, S_i \in C_a$, and O is the super-category of S_i, (i=1..n);
- C_c is a finite set of concrete category names, $C_c \sqsubseteq \widehat{C}_c$;
- R is a finite set of relations in the form of $r(c{:}\ t :: d \rightarrow f)$, where $r \in \widehat{R}$, $c, t \in \widehat{V}$, $d \in C_a$, $f \in C_a \cup C_c$ and c denotes the cardinality of r , t the totality, d the domain, and f the range;

For example, the Publication Semantic Schema in Figure 2 can be formalized as $\mathcal{H}_1 = (C_a, W, C_c, R)$, where:

- C_a=*{PUBLICATION, BOOK, CONTACTAUTHOR, ARTICLE, AUTHOR, NAME}*
- W =*{(PUBLICATION, BOOK, ARTICLE)}*
- C_c =*{String}*

[1] A relation R whose domain is C is total if at all times, for every object x in category C, there exists an object y such that xRy.

– R ={*title(m_1:total::PUBLICATION* $\rightarrow$ *String),*
adress(m_1:total::AUTHOR $\rightarrow$ *String),*
id(m_1: total::AUTHOR $\rightarrow$ *String),*
first(m_1:not_total::NAME $\rightarrow$ *String),*
last (m_1:total::NAME $\rightarrow$ *String),*
book_author(m_1:total::BOOK $\rightarrow$ *AUTHOR),*
article_author(m_m:not_total::ARTICLE $\rightarrow$ *AUTHOR),*
the_contactauthor(m_1:total::ARTICLE $\rightarrow$ *CONTACTAUTHOR),*
author_name(m_1:total::AUTHOR $\rightarrow$ *NAME),*
contact_author(m_1:not_total: CONTACTAUTHOR $\rightarrow$ *AUTHOR)}*

4. TRANSFORMATION FROM A SEMANTIC SCHEMA TO A DTD

Converting a Semantic Schema to a DTD is straightforward, compared to converting a relational schema to a DTD. There are two basic constructs in DTDs, elements and attributes. The relationships among elements can be represented by either parent-child element relationships or ID and IDREF/IDREFS attribute pairs. We present the mapping rules and algorithm below.

Intuitively, we can map categories to elements, and relations to parent-child element relationships or ID and IDREF/IDREFs pairs. Sometimes both mapping alternatives are correct, while one might be better and more semantically meaningful than the other in other situations. Attributes can be mapped to either elements or attributes according to the circumstances. However, there are still some subtle points which need special attention in the transformation process, for example, how to deal with inheritance, etc.

The detailed mapping rules from a Semantic Schema $\mathcal{H}=(C_a, W, C_c, R)$ to a DTD $\mathcal{G} = (E, A, S, P)$ are described as follows.

1) Category mapping:
   Map each abstract category $C \in C_a$ to an element E_c, i.e., $E = E \cup \{ E_c \}$. For example, Categories *BOOK* and *ARTICLE* in Figure 2 are mapped as elements *BOOK* and *ARTICLE*, respectively.

2) Attribute mapping:
   For each relation $r \in R$, where $r(c: t ::d \rightarrow f)$, $c, t \in V$, $d \in C_a, f \in C_c$ (i.e. r is an *attribute* of d)

   a) If c = 1_1 or m_1

      i) If f = Enumerate

         Then r is mapped to an attribute a_r of ENUM type of E_d, where E_d is the element corresponding to category d, i.e. E_d (a_r: ENUM: d: v), and the default type d is determined as follows:

         - If t = total, then d = *REQUIRED*
         - else, $d = \epsilon$

         Additionally, the default value v is determined by the default value of the relation r.

      ii) Otherwise (r is not of ENUM type)

         Then r is mapped to an attribute a_r of CDATA type of E_d, where E_d is the element corresponding to category d, i.e. E_d (a_r: *CDATA*: d: v), and default type d and default value of the attribute are determined in same procedure as the above.

         For example, Category *PUBLICATION* in Figure 2 has a total attribute called *title* which is of String type. It will be mapped as a REQUIRED attribute of the element *PUBLICATION*, i.e., we will have *PUBLICATION (title: CDATA: REQUIRED:* ϵ*).*

   b) If c = 1_m or m_m

      Then r is mapped to a sub-element E_r of E_d, where E_d is the element corresponding to the category d, $E_d \rightarrow (...E_r...)$ and $E_r \rightarrow PCDATA$. The cardinality of E_r in E_d is determined as follows:

      i) If t =not_total, then $E_d \rightarrow (... E_r{*}...)$
      ii) If t =total, then $E_d \rightarrow (... E_r{+}...)$

      In the above mapping algorithm, a *1:m* or *m:m* attribute is mapped as an element instead of an attribute. This is because in DTDs no attribute type except the IDREFS type can express the *1-to-m* multiplicity. Since in some situations the IDREFS type is not appropriate for this transformation, a general solution is to map *1:m* or *m:m* attributes to sub-elements of their domain elements, as shown above.

      Note that because Sem-ODM supports *1:m* attributes, we don't have to go through the nesting process that was proposed in the NeT algorithm [5], where scanning the entire table has to be done in order to find out the multi-valued attributes. A *1:m* attribute is naturally converted to a sub-element of a parent element corresponding to the category that this attribute belongs to.

3) Relation Mapping:
   For each relation $r \in R$, where $r(c: t:: d \rightarrow f)$, $c, t \in V$, $d \in C_a$, $f \in C_a$ (i.e. r is a *relation* between two abstract category d and f)

   Then r is mapped to the sub-element relationship between E_d and E_f, and $E_d \rightarrow (.....E_f....)$, where E_d and E_f are elements corresponding to the abstract category d and f. The cardinality of E_f in E_d is determined as follows:

   i) If c = m_1 or 1_1, and t = not_total,
      then $E_d \rightarrow (... E_f?...)$
   ii) If c = m_1 or 1_1, and t = total, then $E_d \rightarrow (... E_f...)$
   iii) If c = m_m or 1_m, and t = not_total,
      then $E_d \rightarrow (... E_f{*}...)$
   iv) If c = m_m or 1_m, and t = total,
      then $E_d \rightarrow (... E_f{+}...)$

In the above transformation, we put the range element directly as a sub-element of the domain element, for instance, the relation *book_author* with domain *BOOK* and range *AUTHOR* is mapped as the sub-element relationship between *BOOK* and *AUTHOR* as in *BOOK* $\rightarrow$ *(AUTHOR)*. This is because a relation in a Semantic Schema actually indicates the relationship between the domain category and range category. In a DTD, such a relationship is embodied by the sub-element relationship between the parent element (corresponding to the domain) and child element (corresponding to the range). Hence, it is not necessary to keep the relation name in the

DTD. However, in some situations, it may be desirable to keep the relation name to indicate the semantics of the sub-elements. For instance, suppose there are two relation r_1 and r_2 from category *d* to category *f*. We would have the following mapping result if we followed the above mapping scheme: $E_d \rightarrow (f^c, f^{c'})$, where *c* and *c'* represent the mapped cardinality for r_1 and r_2, respectively, and are determined according to the above description. It's not clear to users what these two *f*s represent. A better solution is to transform such relations into $E_d \rightarrow (r_1^c, r_2^{c'})$, $r_1 \rightarrow E_f$ and $r_2 \rightarrow E_f$. Therefore, to make the mapping semantically easier to understand, we let the designer to tune the DTD at the end of the transformation. In this way, they can introduce appropriate intermediate elements which can correctly represent the semantics of the sub-element relationship.

Note that we mentioned at the beginning of the section that the relations in Sem-ODM can also be mapped to ID and IDREF/IDREFS pairs in DTD. Though it is possible, we feel that it reduces the level of nesting in the resulting DTD. Thus, we adopt the sub-element approach and leave it to the users to decide whether or not to tune the mapping at the end.

4) Inheritance mapping

For each $u = (O, S_1, S_2, \ldots S_n) \in W$ in $\mathcal{H}$, where $O, S_i \in C_a$, (*i*=1..n), and *O* is the super-category of S_i, create an attribute *id* of ID type with #REQUIRED default type in E_O and an attribute *id* of IDREF type with #REQUIRED default type in each E_{S_i}, where E_O is the element corresponding to super-category *O* and E_{S_i} is the element corresponding to S_i (*i*=1..n).

For example, *PUBLICATION* is the super-category of *BOOK* and *ARTICLE* in Figure 2, we map this inheritance relationship to

PUBLICATION $\rightarrow (\epsilon)$,
PUBLICATION(id:ID:REQUIRED:ϵ),
BOOK(id:IDREF:REQUIRED:ϵ)
ARTICLE(id:IDREF:REQUIRED: ϵ).

The reasoning behind using an ID and IDREF pairs instead of a sub-element relationship is ID and IDREF pairs have the semantic of "is-a" while sub-element relationships have the "has" semantic. While it is obvious, inheritance between sub-categories and super-categories exhibits an "is-a" relationship.

5) Determine the root element set *S*

Since any element in the DTD can become a root element, we set *S* = *E*.

The mapping algorithm from a Sem-ODM Semantic Schema $\mathcal{H} = (C_a, W, C_c, R)$ to a DTD $\mathcal{G} = (E, A, S, P)$ is described as follows:

1. For each unmapped abstract category $C \in C_a$, create an element *C* in *E*, then
   - Check if it has super-category, if it does, follow the above rule 4)
   - For each attribute $att \in R$, follow the above rule 2) for Attribute Mapping
   - For each relation $rel \in R$, follow the above rule 3) for Relation Mapping using the depth-first search to map all the categories that are reachable from *C*
   - Mark this category as mapped
2. Repeat the above step 1 until all the categories are marked
3. Set *S=E*

At the end of the mapping process, users can tune the resulting DTD into one that better expresses the semantics by, for example, using a relation name representing the parent-child relationship as explained in the rule 3 above or changing the sub-element relationship mapping to IDREF reference mapping. For instance, in Figure 2 the relation *contact_author* between category *CONTACTAUTHOR* and *AUTHOR* is mapped to the sub-element relationship between *CONTACTAUTHOR* and *AUTHOR* as in *CONTACTAUTHOR* $\rightarrow$ *(AUTHOR?)* since *contact_author* is *m:1*, *not_total*. An alternative way to express this relationship is to create an attribute *id* of ID type with #REQUIRED default type in *AUTHOR* (omitted in our example, since there is already such an attribute) and then create an attribute *ref_author* of IDREF type with #IMPLIED default type in *CONTACTAUTHOR*, i.e.,

AUTHOR(id: ID: REQUIRED: ϵ)
CONTACTAUTHOR(ref_author: IDREF: IMPLIED: ϵ)

In case *contact_author* is *1:m* or *m:m* (no matter what totality it has), an IDREFS, instead of IDREF, attribute is created in *CONTACTAUTHOR*. If *contact_author* is *total* (no matter what cardinality it has), then the attribute *ref_author* of IDREF type (if with m:1 or 1:1 cardinality) or IDREFS type (if with m:m or 1:m cardinality) has #REQUIRED default type in *CONTACTAUTHOR*.

For example, the DTD corresponding to the *Publication* Schema in Figure 2 is $\mathcal{G}_2 = (E, A, S, P)$, where:

- *E*= {PUBLICATION, BOOK, ARTICLE, AUTHOR, CONTACTAUTHOR, NAME }
- *A*= {PUBLICATION(title: CDATA: REQUIRED: ϵ), PUBLICATION(id: ID: REQUIRED: ϵ), BOOK(id: IDREF: REQUIRED: ϵ), ARTICLE(id:IDREF:REQUIRED:ϵ), AUTHOR(address:CDATA:REQUIRED:ϵ), AUTHOR(id: ID: REQUIRED: ϵ), NAME(first: CDATA: ϵ: ϵ), NAME(last: CDATA: REQUIRED: ϵ)}
- *S* ={PUBLICATION, BOOK, ARTICLE, AUTHOR, CONTACTAUTHOR, NAME }
- *P* ={PUBLICATION $\rightarrow$ (ϵ), BOOK $\rightarrow$ AUTHOR),
  ARTICLE $\rightarrow$ (AUTHOR*, CONTACTAUTHOR),
  AUTHOR $\rightarrow$ (NAME), NAME $\rightarrow$ (ϵ)
  CONTACTAUTHOR $\rightarrow$ (AUTHOR?) }

Figure 3 shows the typical DTD representation of the above formal DTD representation.

Note that Data in traditional databases is often regarded as un-ordered. This characteristic also holds in the Sem-ODM. However, this is not the case for the XML data model. For instance, in the DTD example in Figure 1, there is an order between *title* and *author*: the *title* must appear before *author* in a *book*. This ordering concept is expressed by the concatenation operator (,) between *title* and *author* in the content model of element *book*. Such an order is sometimes called the *Element Order* [7]. When translating a Semantic Schema into a DTD, an implicit ordering is created according to the order of processing. For example, in our example, category *ARTICLE* and its two relations *article_author* and *the_contactauthor* are translated into *ARTICLE* $\rightarrow$ *(AUTHOR*, CONTACTAUTHOR)*, where *AUTHOR* is listed as the first element of *ARTICLE*. It's also possible that *CONTACTAUTHOR* is listed first depending on which relation gets translated first. Again, the designer can tune the ordering as it fits best.

```
<!ELEMENT publication EMPTY>
<!ATTLIST publication   id ID #REQUIRED>
<!ATTLIST publication   title CDATA #REQUIRED>
<!ELEMENT book (author)>
<!ATTLIST  book  id IDREF #REQUIRED>
<!ELEMENT article (author*, contactauthor)>
<!ATTLIST article   id ID #REQUIRED>
<!ELEMENT contactauthor (author?)>
<!ELEMENT author (name)>
<!ATTLIST author   id ID #REQUIRED>
<!ATTLIST author   address CDATA #REQUIRED>
<!ELEMENT name EMPTY>
<!ATTLIST  name  first CDATA>
<!ATTLIST  name  larst CDATA #REQUIRED>
```

Figure 3. Generated DTD

The mapping algorithm we present here has several properties. First, it's lossless in the sense that all the structure (such as categories and attributes) and semantic information (such as relationships between categories) are kept in the resulting DTD. Second, the resulting DTD has desirable nested structure. We can prove the correctness of the algorithm. But due to space constraints, it is not provided here. The complexity of the algorithm is $\Theta(N+M)$ where N is the total number of categories (vertices) and M is the total number of relations including attributes (edges) in the Semantic Schema.

5. CONCLUSION AND FUTURE WORK

In this paper, we formally described DTDs and Sem-ODM Semantic Schemas in order to facilitate the explanation of schema transformation between the two data models. A Sem-ODM Semantic Schema can be converted into a DTD by mapping categories to elements, attributes to attributes or elements, and relations to parent-child element relationships. The differences, similarities, and possible connection of the two data models are expressed via the formal representation of the two schemas.

We are working on extending this work to Semantic Schemas to XML Schemas mapping and the implementation. Compared to XML Schema, DTD has some limitations, such as very limited data types, untyped IDREF(S), among others. Some data type information in the Semantic Schema is lost during the Semantic Schema to DTD transformation process because DTD is geared toward the support of String data. Some features of the Sem-ODM cannot be exploited when mapping Semantic Schemas to DTDs. For example, the Sem-ODB supports user-defined Integer, Real, Enumerate, and String categories while providing a way to specify the data format for each type. Users can specify the minimum and maximum number of an Integer Category, or use regular expressions to denote the format of a String category. Similar features are supported in XML Schema. We expect to be able to utilize these rich semantic features of the Sem-ODM in publishing Sem-ODB data as XML.

6. ACKNOWLEDGEMENTS

This research was supported in part by NSF Grants EIA-0320956, EIA-0220562, and HRD-0317692.

7. REFERENCES

[1] T. Bray, J. Paoli, C. M. Sperberg-McQueen and E. Maler (Eds), "Extensible Markup Language (XML) 1.0 (Second Edition)", W3C Recommendation, October 2000, http://www.w3.org/TR/2000/REC-xml-20001006.

[2] Diego Calvanese, Giusepe D. Giacomo, Maurizio Lenzerini, "Representing and Resaoning on XML Documents: A Description Logic Approach", Journal of Logic and Computation, Vol. 9, No. 3, pp 295-318, Oxford University Press, 1999.

[3] David C. Fallside (Ed), "XML Schema Part 0: Primer", W3C Recommendation, May 2 2001, http://www.w3.org/TR/xmlschema-0/.

[4] Dongwon Lee, Murali Mani, Frank Chiu, Wesley W. Chu, "Net & CoT: Translating Relational Schemas to XML Schemas using Semantic Constraints", CIKM'02.

[5] Dongwon Lee, Murali Mani, Frank Chiu, Wesley W. Chu, "NeT & CoT: Inferring XML Schemas from Relational World", Proc. of (ICDE), San Jose, CA, USA, 2002.

[6] Murali Mani, Dongwon Lee, Richard R. Muntz, "Semantic Data Modeling using XML Schemas", Proc. 20th Int'l Conf. on Conceptual Modeling (ER), Yokohama, Japan, 2001.

[7] Ioana Manolescu, Daniela Florescu, and Donald Kossmann, "Pushing XML queries inside relational databases", Tech. Report No. 4112, INRIA, January 2001.

[8] Makoto Murata, Dongwon Lee, Murali Mani, "Taxonomy of XML Schema Languages using Formal Language Theory", Extreme Markup Languages, Montreal, Canada, 2000.

[9] M. Murata, "RELAX (Regular Language description for XML)", Aug. 2000, http://www.xml.gr.jp/relax

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

[11] Naphtali Rishe, "TERRAFLY: A High-Performance Web-based Digital Library System for Spatial Data Access", ICDE'01 (Demo), pp17-19, 2001.

[12] Michael Rys, "Bringing the Internet to Your Database: Using SQLServer 2000 and XML to Build Loosely-Coupled Systems", In Proceedings of ICDE 2001, pp. 465-472, Heidelberg, Germany, April 2001.

[13] Jayavel Shanmugasundaram, Kristin Tufte, Gang He, Chun Zhang, David DeWitt, Jeffrey Naughton, "Relational Databases for Querying XML Documents: Limitations and Opportunities", Proc. of VLDB, Edinburgh, UK, 1999.

[14] Victor Vianu, "A Web Odyssey: from Codd to XML", Proceedings of the 20th ACM SIGACT-SIGMOD-SIGART Symposium on (PODS), Santa Barbara, CA, USA, 2000.