Representation and Transformation of DTDs and Semantic Schemas: A Formal Framework

Naphtali Rishe, Li Yang, Maxim Chekmasov, Scott Graham

High Performance Database Research Center

School of Computer Science

Florida International University

ECS243, 11200 S.W. 8th Str.

Miami, FL 33199-0001

Tel: (305) 348-1706 Fax:(305) 348-1705

Email:{rishen, lyang01, maximc, grahams}@cs.fiu.edu

Abstract

Many projects have investigated the issue of storing XML in traditional database systems and exporting data in traditional databases as XML documents. However, they have paid little attention to the formalization of the data models and transformation between them. In this paper, we address this problem by providing a formal framework in which Document Type Definitions and the Semantic Binary Object-Oriented Data Model's Semantic Schemas are formally defined and converted into each other. Through this formalization, we achieve preciseness and conciseness in expressing both data models and performing transformations between them.

Key words: XML, DTD, Sem-ODM, Semantic Schema, Data Model Transformation.

1. Introduction

With the popularity of XML (eXtensible Markup Language) [1] as a data exchange and representation format on the Web growing, many XML-related issues have been studied in different projects; for example

[9, 5, 21,7, 10] focused on the storage issues of XML and [2, 8] investigated issues in publishing data stored in traditional databases as XML. However, to the best of our knowledge, none of the projects that transform XML into traditional database models (relational, object-oriented, or object-relational) or vice-versa formalized their work, even though formalization is a very important mechanism for understanding the data models and the transformations between them.

This paper is based on our work in [20], in which we described in detail the mapping from a DTD (Document Type Definition) [1] to a Semantic Schema of Sem-ODM (Semantic Binary Object-Oriented Data Model) [18] in order to store XML in Sem-ODB (Semantic Binary Object-Oriented Database System). In this paper, we study the formalization aspect of the transformation between the schemas of two data models, XML and Sem-ODM. We first present formal definitions of a DTD and a Semantic Schema. We then describe a formal framework in which a DTD and a Semantic Schema are converted into each other. The formalization of the transformation connects the two data models and helps us better understand the capabilities brought about by storing XML in Sem-ODB and exporting data in Sem-ODB as XML.

The following contributions are made by this paper:

- We formalize the XML schema language DTD and the Sem-ODM Semantic Schema. By doing so, we gain a better and precise understanding of the schemas of these two different data models.
- We present the schema transformation between a DTD and a Semantic Schemas formally in both directions, i.e., DTD to Semantic Schema and vice-versa. In this way, we describe the possible relationships and conversions between these two data models.

1.1. Related Work

Efficiently storing and querying XML data has been a research issue studied in various projects [7, 5, 21, 22, 10, 11, 6, 9, 23]. From the perspective of underlying data storage mechanisms adopted, this research work can be classified into three categories: the relational database approach [7, 5, 21, 22, 10, 11], the object-oriented database approach [6], and the native XML storage approach [9, 23]. The relational database approach has received a great deal of study due to its mature technological base and its widely market-dominating commercial implementations. Most research work using this approach first creates a

relational schema to store the structure and constraints of the XML data, and then loads the data into the resulting databases. Projects adopting this approach can be further categorized according to whether or not they utilize an XML schema (e.g., DTD, XML Schema [26]) for relational schema generation. The first category, schema-based conversion, requires a DTD/XML Schema available during the relational schema generation phase. [21, 10, 11] are examples of this category. Another category, schema-less conversion, does not rely on DTDs/XML Schemas to create the relational schemas. [7, 5, 22] fall into this category. Tian et al. [24] evaluates the performance of six storage alternatives (one using file system, three using relational databases, and two using an object manager), with both schema-based and schema-less conversions involved. Our approach of generating a Semantic Schema for XML documents is a schema-based conversion. We require the existence of a DTD before performing the schema transformation.

Another research direction, publishing data in traditional databases into XML, has also been studied. Lee et al. [12, 13] devises two algorithms to convert a single relational table or multiple interconnected relational tables into DTDs based on inclusion dependencies that are obtained from the underlying relational databases. In the XPERATO project [2], the authors first map the underlying object-relational database schema into a default XML view whose structure is described by XML Schema. This mapping is more intuitive than the mapping from a relational schema to a DTD, because XML Schema is designed in such a way that it richly expresses constraints, semantics, and data types. In this paper, we address the issue of mapping a Sem-ODM Semantic Schema to a DTD, not an XML Schema, because of the popularity of DTDs. Unlike [12, 13], we do not rely on inclusion dependencies to generate the hierarchical structure of XML. Because of the explicit and expressive relations between objects in the Sem-ODM, our conversion can be performed in a much simpler way than the one in [12, 13]. Additionally, we describe the conversion of a database schema, which involves the interleave of multiple categories (similar to tables in the relational model), not just a single category as in [12].

Several XML schema languages, such as DTD and XML Schema among others, have been proposed to describe the structure and semantics of XML documents. [15] formally described several XML schema languages (DTD, XML Schema, RELAX [17], and others) based on regular tree languages. It represented a DTD as a local regular tree grammar, whereas [25] represented a DTD as an extended context free grammar and [3] presented a DTD in terms of Description Logic. Lee et al. [12] formalized relational

schemas and DTDs, and presented a nesting-based translation algorithm to transform a relational schema into a DTD. In [16], 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 [12] and has been influenced by the formalism presented in [16]. However, [16] 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 transformations between a DTD and a Sem-ODM Semantic Schema, not relational schemas. We also present transformations for both directions, DTD to Semantic Schema and vice-versa, not just one direction; in this sense, our work is more complete.

1.2. Outline of the paper

In the rest of this paper, we first present the formal definitions of a DTD and a Semantic Schema in section 2. Section 3 formally describes the mapping from a DTD to a Semantic Schema. The formal description of mapping from a Semantic Schema to a DTD is presented in section 4. Section 5 concludes this paper.

2. Formal Definitions of DTD and Semantic Schema

2.1. Definition of DTD

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 [16, 3, 12]. We do not consider ENTITY, ENTITIES, NMTOKEN, and NMTOKENS attribute types in this paper.

Before we proceed to define DTDs, we first make some assumptions on some notations. 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 where $\hat{\tau}$ ::= {CDATA,ENUM, ID, IDREF, IDREFS}, $\hat{d}$ is a set of default types that are

allowed in a DTD attribute and where $\hat{d}$ ::={IMPLIED, REQUIRED, FIXED} or ϵ which represents the case where no default type is specified, and that $\hat{u}$ is the default value of an attribute where $\hat{u}$= a set of strings or integers allowed in a DTD or $\hat{u}=\epsilon$ (representing no default value) if 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: τ: 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 $X \rightarrow r$, where $X, Y \in E$ and *r* is the content model of *X* and can be generalized in the following abstract syntax:

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

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 [21] 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) }

Note that, as in our previous work in [20], we simplify the DTD before the mapping process so that the DTD does not contain the Choice content type. Hence, in the following sections, we ignore $r|r$ in the definition of r.

2.2. Definition of Semantic Schema

The Sem-ODM (Semantic Binary Object-Oriented Data Model) is a high-level data model which was developed at the High-Performance Database Research Center (HPDRC) [19]. As a conceptual level data model, it can mirror the real world enterprise scenarios naturally as the ER (Entity Relationship) model does. In addition, it has some advantages of the Object-Oriented data model, such as inheritance, oids, and explicit relationships among objects, etc.

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, for example, categories 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 name 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 *the_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 $\hat{C}_a$ is a finite set of abstract category names, $\hat{C}_c$ is a finite set of concrete category names, $\hat{R}$ is a finite set of relation names, and $\hat{V}$ is a finite set of strings representing the values of cardinality and totality.

(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, $C_a \subseteq \hat{C}_a$;
- 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 categories name, $C_c \subseteq \hat{C}_c$;
- R is a finite set of relations in the form of $r(c: t :: d \rightarrow f)$, where $r \in \hat{R}$, c, $t \in \hat{V}$, $d \in C_a$, $f \in C_a \cup C_c$ 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, ARTICLE, AUTHOR, CONTACTAUTHOR, NAME}
- W ={(PUBLICATION, BOOK, ARTICLE)}
- C_c ={String}
- R={title(m_1[2]:total::PUBLICATION $\rightarrow$ 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.
[2] m_1 represents cardinality m:1, and similarly 1_1, 1_m and m_m represent 1:1, 1:m and m:m, respectively.

address(m_1: total::AUTHOR ⟶ String), id(m_1: total::AUTHOR ⟶ String),

first (m_1:not_total::NAME ⟶ String), last (m_1:total::NAME ⟶ String),

the_author(m_1: total::BOOK ⟶ AUTHOR),

the_author(m_m:not_total::ARTICLE ⟶ AUTHOR),

the_contactauthor(m_1: total::ARTICLE ⟶ CONTACTAUTHOR),

author_name(m_1: total::AUTHOR ⟶ NAME),

the_author(m_1:not_total::CONTACTAUTHOR ⟶ AUTHOR)}

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* [14]. In addition, there is another more important ordering concept in the XML data model, the *Document Order* [4]. Once an XML document is created, there is a total order among the elements within the documents. For instance, in Figure 1 the element *article* might have multiple *authors*. It is meaningful to differentiate between the first author, the second one, and so on.

It is easy to incorporate an ordering concept to support *document order* in the Sem-ODM. Because the Sem-ODM supports user-defined object identifiers (uids) for abstract and binary objects, we can ensure a total order among these objects through uniquely defined uids. As for the element order, once the document order is supported, it does not seem to be that important, though we still preserve such information in our mapping meta-schema. Interested readers can refer to [20] for more details.

3. Transformation from a DTD to a Semantic Schema

The basic constructs of a DTD are elements and attributes. Therefore, the mapping is considered from two perspectives: element-related and attribute-related. The basic idea of our mapping algorithm (see [20]) is to map the majority of elements into categories. Some special elements (e.g. ANY and PCDATA) are

mapped into categories if they do not have any parent element, are shared by multiple parent elements, or appear in their only parent element multiple times, and are otherwise mapped into attributes. The insight here is to inline a sub-element as an attribute of its parent element if it does not appear in its parent element multiple times to reduce the number of categories created. The attributes in a DTD are mapped as relations in a Semantic Schema. Additionally, we map the relationships between sub-elements and their parent elements to relations of two categories corresponding to the elements in DTD.

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

1. Elements of Empty, Sequence, Mixed content.

   For any X in

   a) $X \rightarrow r$, where $X \in E$ and $r ::= \epsilon \mid (r)$, i.e. X is of Empty content, or

   b) $X \rightarrow r$, where $r ::= Y \mid r,r \mid r? \mid r* \mid r+$, and $X, Y \in E$, i.e. X is of Sequence content, or

   c) $X \rightarrow r$, where $r ::= (PCDATA \mid Y)*$, and $X, Y \in E$, i.e. X is of Mixed content

   it is mapped to an abstract category with name X. For example, element *book* in Figure 1, which is a Sequence content element, is mapped to a category called *book* after applying the above rule.

2. Elements of PCDATA content.

   For any X in $X \rightarrow r$, where $r ::= (PCDATA) \mid (PCDATA)*$, and $X \in E$, i.e. X is of PCDATA content,

   a) if X has only one parent element, Y, and X appears in Y with cardinality only one or '?',

      i.e. $Y \rightarrow r$, where $Y \in E$, X appears in r and no $Z \in E$ exists such that $Z \rightarrow r'$, X appears in r', and $Y \rightarrow (….X….)$ or $Y \rightarrow (….X?….)$

      then inline X as an attribute of Y, i.e., map X to an attribute called X such that $X(c:t::Y \rightarrow String)$, where c, and t are determined by the following:

      i. If $Y \rightarrow (….X….)$, and $X \rightarrow (PCDATA)$, then c = m_1, t = total

      ii. If $Y \rightarrow (….X….)$, and $X \rightarrow (PCDATA)*$, then c = m_1, t = not_total

      iii. If $Y \rightarrow (….X?….)$, then c = m_1, t = not_total

For example, consider *first* in Figure 1. It is an example of a). It will be mapped to an attribute of category *name*, such that *first(m_1:not_total::name ⟶ String).*

Note that in this case, any attribute *a* of *X* has to be mapped as an attribute of *Y* with name *X_a* (see rule 5 for more detail).

b) if *X* does not have parents or has more than one parent,

Then *X* is mapped to a Category called *X* with one attribute *r* such that *r(c:t::X ⟶ String)*, where *c*, and *t* are determined as follows:

i. If *X* ⟶ *(PCDATA)*, Then *c* = m_1, *t* = total

ii. If *X* ⟶ *(PCDATA*)*, Then *c* = m_m, *t* = not_total

For example, consider the element *title* in Figure 1. It will be mapped as a category *title* with an attribute *data*, as *data(m_1: total: title ⟶ String).* In addition, since it has two parent elements, *book* and *article*, the parent-child relationships in $\mathcal{G}$ are mapped as two relations in $\mathcal{H}$, one from *book* to *title* and the other from *article* to *title*, as *book_title(m_1: total::book ⟶ title)* and *article_title(m_1: total::article ⟶ title)* according to the rule 4 below.

Or

c) if *X* has one parent *Y* but the cardinality of *X* in *Y* is '*' or '+', then

X is mapped to a Category *X* as explained in b) above.

3. Elements of ANY content.

For an element *X* of ANY content

a) if *X* has only one parent element *Y*, and *X* appears in *Y* with cardinality only one or '?', *X* can be handled in the same way as 2-a)-i and 2-a)-iii above, except that we need to modify the range of relation *X* from *String* to *XMLType*, which is a special data type that is introduced in the Sem-ODM to handle data of ANY type. For example, element *address* is mapped to an attribute of *author* as *address(m_1: total::author ⟶ XMLType).*

b) if X does not have parents or has more than one parent, *X* is mapped to a Category called *X* with one attribute *r* such that $r(m\_1:not\_total::X \rightarrow XMLType)$.

c) if *X* has one parent *Y* but the cardinality of *X* in *Y* is '*' or '+', then it is treated the same as in 3-b).

4. Relationship mapping

a) Relationship in Sequence content.

For every *X* in the form of $X \rightarrow (r_1, r_2, \ldots r_n)$, where $r_i ::= Y_i \mid Y_i ? \mid Y_i * \mid Y_i +$ and $Y_i \in E$ (i=1..n), we map the sub-element relationship between *X* and r_i into a relation $X\_r_i\ (c:t::X \rightarrow Y_i)$, where *c* and *t* are determined as follows:

i. If $r_i ::= Y_i$, then *c* = m_1, *t* = total

ii. If $r_i ::= Y_i?$, then *c* = m_1, *t* = not_total

iii. If $r_i ::= Y_i*$, then *c* = m_m, *t* = not_total

iv. If $r_i ::= Y_i+$, then *c* = m_m, *t* = total

For example, the sub-element relationship between *book* and *author* in Figure 1 is mapped to a relation called $book\_author\ (m\_1:\ total::book \rightarrow author)$.

b) Relationship in Mixed content.

Similarly, for every *X* in the form of $X \rightarrow (PCDATA \mid Y_1 \mid \ldots \mid .Y_n)*$, where $Y_i \in E$ (i=1..n), each sub-element relationship between *X* and Y_i can be mapped into a relation described in 4-a)-iii. As for the *PCDATA* part, a category with a system-generated unique name *C* will be created with an attribute *r* such that $r(m\_1:total::C \rightarrow String)$. Then a relation will be created for *X* and *C* as $X\_C\ (m\_m:not\_total::X \rightarrow C)$.

5. Attribute mapping

For each item $u \in A$, where $u = X(a{:}\tau{:}d{:}v)$, where $X \in E$, $a \in \hat{A}$, $\tau \in \hat{\tau}$, $d \in \hat{d}$, $v \in \hat{u}$, we map attribute *a* to a relation and the property of the relation is determined according to the following rules:

(1) If *X* falls into 2-a), i.e., *X* is of PCDATA content element with one parent element *Y* and *X* appears in *Y* with only one or ? cardinality, then *X* is inlined as an attribute of *Y* in this case in 2-a). In this situation, we have to inline all the attributes of *X* as attributes of *Y*, i.e., we map attribute *a* to a relation $X\_a(c{:}t{::}C_Y \rightarrow E)$, where C_Y denotes the category corresponding to *Y*, and *c*, *t* and *E* are defined as follows.

   a) If $Y \rightarrow (\ldots X \ldots)$, then *c*, *t* and E are determined in the same procedure as in the following (2).

   b) If $Y \rightarrow (\ldots X? \ldots)$, then *t* = not_total and *c* and *E* are determined in the same procedure as in the following (2).

(2) Otherwise, we map attribute *a* to a relation *a* of *X*, i.e., $a(c{:}t{::}C_X \rightarrow E)$, where C_X denotes the category corresponding to element *X* and *c*, *t* and *E* are defined as the following:

   a) If τ = CDATA, then

      i. If *d* = IMPLIED or ϵ, then *c* =m_1, *t* =not_total, *E* =*String*

      ii. If *d* = REQUIRED, then *c* =m_1, *t* =total, *E* =*String*

   b) If τ = ENUM, then

      i. If *d* = IMPLIED or ϵ, then *c* =m_1, *t* =not_total, *E* =*Enumerate* and *v* will be the enumerated values

      ii. If *d* = REQUIRED, then *c* =m_1, *t* =total, *E* =*Enumerate* and *v* will be the enumerated values

   c) If τ = ID, then

      i. If *d* = IMPLIED or ϵ, then *c* =m_1, *t* =not_total, *E* =*String*

      ii. If *d* = REQUIRED, then *c* =m_1, *t* =total, *E* =*String*

      For example, attribute *id* of element *author* in Figure 1 is transformed into an attribute of Category *author*, i.e., $id(m\_1{:}\ total{::}author \rightarrow String)$.

   d) If τ = IDREF, then

i. If d = IMPLIED or ϵ, then c =m_1, t =not_total, E = undefined

ii. If d = REQUIRED, then c =m_1, t =total, E = undefined

Note that in this case, the range category E has to be decided by the designer who is performing the mapping. For example, attribute *authorID* of element *contactauthor* in Figure 1 is transformed into a relation from Category *contactauthor* to Category *author*, i.e., *authorID (m_1: not_total::contactauthor ⟶ author).*

e) If τ = IDREFS, then

i. If d = IMPLIED or ϵ, then c =m_m, t =not_total, E = undefined

ii. If d= REQUIRED, then c =m_m, t =total, E = undefined

Note that similar to case d) when τ = IDREF, the range category E has to be decided by the designer who is performing the mapping.

For example, the DTD in Figure 1 will be mapped to the following Semantic Schema $\mathcal{H}$ = (C_a, W, C_c, R), where:

- C_a={publication,book, author, title, article, name, contactauthor }
- $W = \emptyset$
- C_c ={String, XMLType}
- R={publication_book(m_m:not_total:publication ⟶ book),

  publication_article(m_m:not_total:publication ⟶ article),

  data(m_1: total: title ⟶ String),

  book_title(m_1: total::book ⟶ title),

  book_author(m_1: total::book ⟶ author),

  author_name(m_1:total::author ⟶ name),

  address(m_1: total::author ⟶ XMLType),

  id(m_1:total::author ⟶ String),

  first (m_1: not_total::name ⟶ String),

  last (m_1:total::name ⟶ String),

  article_title(m_1: total::article ⟶ title),

article_author(m_m: not_total::article ⟶ author),

article_contactauthor(m_1: total::article ⟶ contactauthor),

authorID(m_1:not_total::contactauthor ⟶ author)}

4. Transformation from a Semantic Schema to a DTD

Converting a Semantic Schema to a DTD is straightforward, compared to the opposite transformation. We can map categories to elements and relations to parent-child element relationships. Attributes can be mapped to either elements or relations 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, and how to choose the root element, etc.

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

1. Category mapping:

   Map each 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 $r(c: t :: d \longrightarrow f)$, $c, t \in V$, $d \in C_a$, $f \in C_c$ (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, $d = c$

         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: c).*

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 $E_d \rightarrow (\ldots E_r{*}\ldots.)$

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

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.

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:

a) If *c* = m_1 or 1_1, and *t* = not_total, then $E_d \rightarrow (\ldots E_f?\ldots.)$

b) If *c* = m_1 or 1_1, and *t* = total, then $E_d \rightarrow (\ldots E_f\ldots.)$

c) If *c* = m_m or 1_m, and *t* = not_total, then $E_d \rightarrow (\ldots E_f{*}\ldots.)$

d) If *c* = m_m or 1_m, and *t* = total, then $E_d \rightarrow (\ldots E_f{+}\ldots.)$

In the above transformation, we put the range element directly as a sub-element of the domain element, for instance, the relation *the_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 will have the following mapping result if we follow the above mapping scheme: $E_d \rightarrow (f^c, f^c)$, where *c* represents the mapped cardinality and is 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 users 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.

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

5. Introducing a root element E_{root}

The name of E_{root} is decided by the designer. This element E_{root} will become the root element in $\mathcal{G}$, i.e., $S = E_{root.}$, $E = E \cup \{ E_{root} \}$. For example, to transform the Semantic Schema in Figure 2, we introduce a root element called E_{root} = *PUBs*.

6. Introducing a sub-element relationship $E_{root} \rightarrow (E_{S1}^*, E_{S2}^*, \ldots, E_{Sn}^*)$, where E_{S_i} (i=1..n) is the element corresponding to the category in C_a. For example, we introduce PUBs $\rightarrow$ *(PUBLICATION*,BOOK*,ARTICLE*,AUTHOR*,CONTACTAUTHOR*,NAME*)* for the schema in Figure 2.

At the end of the mapping process, designers 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 *the_author* between category *CONTACTAUTHOR* and *AUTHOR* is mapped to the sub-element relationship between *CONTACTAUTHOR* and *AUTHOR* as in *CONTACTAUTHOR* $\rightarrow$ *(AUTHOR?).* 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*. In case *AUTHOR* has '*' or '+' cardinality in *CONTACTAUTHOR*, then an IDREFS, instead of IDREF, attribute is created in *CONTACTAUTHOR*. If *AUTHOR* has only one or '+' cardinality in *CONTACTAUTHOR*, i.e., *CONTACTAUTHOR* $\rightarrow$ *(AUTHOR) or CONTACTAUTHOR* $\rightarrow$ *(AUTHOR+),* then the attribute *ref_author* of IDREF type 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 = {PUBs, 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 = {PUBs}
- P = {PUBs → (PUBLICATION*, BOOK*, ARTICLE*,AUTHOR*,CONTACTAUTHOR*,

NAME*),

PUBLICATION → (ϵ), BOOK → (AUTHOR),

ARTICLE → (AUTHOR*, CONTACTAUTHOR), AUTHOR → (NAME),

CONTACTAUTHOR → (AUTHOR?), NAME → (ϵ) }

5. Conclusions

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. All of the structure and semantic constraints defined in DTDs can be captured naturally in Sem-ODM Semantic Schemas by the transformation algorithms that we proposed. Elements in DTDs are modeled as categories or inlined as attributes in some special cases, while attributes, and parent-child element relationships are converted into binary relations in the Sem-ODM. A Sem-ODM Semantic Schema can also be converted into a DTD by mapping categories to elements, attributes to attributes or elements, and relations to parent-child element relationships. By formalizing the transformation in both directions, the differences, similarities, and possible interaction of the two data models are expressed.

The implementation of the transformations is ongoing. We have implemented the transformation from DTDs to Semantic Schemas and plan to continue until the mappings in both directions are complete. In addition, we would like to extend our work to XML Schema. 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.

Acknowledgements

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.

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] Michael Carey, Daniela Florescu, Zachary Ives, Ying Lu, Jayavel Shanmugasundaram, Eugene Shekita, Subbu Subramanian, "XPERANTO: Publishing Object-Relational Data as XML", Proceedings of the 26th Int. Conf. On Very Large Data Bases (VLDB), Cairo, Egypt, 2000.

[3] 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.

[4] Denise Draper, Peter Fankhauser, Mary Fernández, Ashok Malhotra, Kristoffer Rose, et al (Eds), XQuery 1.0 Formal Semantics,W3C Working Draft 26 March 2002.

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

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

[8] M. Fernandez, W. Tan, D. Suciu, "SilkRoute:Trading Between Relations and XML", 9th International WWW Conference, Amsterdam, May 2000.

[9] 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.

[10] M. Klettke, H. Meyer, "XML and Object-Relational Database Systems – Enhancing Structural Mappings Based on Statistics", Proceedings of the Third International Workshop on the Web and Databases (WebDB 2000), Dallas, Texas, USA, May, 2000.

[11] Dongwon Lee, Wesley W. Chu, “CPI: Constraints-Preserving Inlining Algorithm for Mapping XML DTD to Relational Schema”, Journal of Data & Knowledge Engineering (DKE), Vol. 39, No 1, page 3 - 25, October 2001.

[12] Dongwon Lee, Murali Mani, Frank Chiu, Wesley W. Chu, “Nesting-based Relational-to-XML Schema Translation”, ACM SIGMOD Int'l Workshop on the Web and Databases (WebDB), Santa Barbara, CA, USA, May 2001.

[13] Dongwon Lee, Murali Mani, Frank Chiu, Wesley W. Chu, "NeT & CoT: Inferring XML Schemas from Relational World", Proc. 18th IEEE Int'l Conf. on Data Engineering (ICDE), San Jose, CA, USA, February, 2002 (Poster).

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

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

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

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

[18] Naphtali Rishe, “Database Design: The Semantic Modeling Approach”, McGraw-Hill, 1992.

[19] N. Rishe, W. Sun, D. Barton, Y. Deng, C. Orji, M. Alexopoulos, L. Loureiro, C. Ordonez, M. Sanchez, A. Shaposhnikov, “Florida International University High Performance Database Research Center”. In SIGMOD Record, Vol. 24, NO. 3, pp. 71-76, 1995.

[20] Naphtali Rishe, Li Yang, Maxim Chekmasov, Marina Chekmasova, Scott Graham, Alejandro Roque, “Mapping from XML DTD to Semantic Schema”, accepted by the 6th World Multiconference on Systemics, Cybernetics and Informatics (SCI), Orlando, FL, USA, July 2002.

[21] Jayavel Shanmugasundaram, Kristin Tufte, Gang He, Chun Zhang, David DeWitt, Jeffrey Naughton, “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.

[22] Takeyuki Shimura, Masatoshi Yoshikawa and Shunsuke Uemura, “Storage and Retrieval of XML Documents using Object-Relational Databases”, 10th International Conference Database and Expert Systems Applications (DEXA '99), Florence, Italy, August 1999.

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

[24] Feng Tian, David J. DeWitt, Jianjun Chen, and Chun Zhang, “The Design and Performance Evaluation of Alternative XML Storage Strategies”, ACM Sigmod Record, Volume 31, Number 1, March 2002.

[25] Victor Vianu, “A Web Odyssey: from Codd to XML”, Proceedings of the 20th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS), Santa Barbara, California, USA, May 2001.

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

Vitae

Naphtali David Rishe

Dr. Rishe's expertise is in database management and Internet. Rishe's methodology for the design of database applications and his work on the Semantic Binary Database Model were published as a book by Prentice-Hall in 1988. Rishe's Semantic Modeling theory was published as a book by McGraw-Hill in 1992. Rishe's current research focuses on efficiency and flexibility of database systems (particularly of object-oriented, semantic, decision-support, and spatial/geographic DBMS), distributed DBMS, high-performance systems, database design tools, and Internet access to databases. Rishe is Editor of 4 books and Author of 3 patents, 24 papers in journals (including IEEE KDE, DKE, Information Systems, Fundamenta Informaticae), 7 chapters in books and serials (including 3 in Springer Verlag's LNCS), 3 encyclopaedia articles, over 80 papers published in proceedings (including ACM SIGMOD, VLDB, PDIS, IEEE DE, ACM SIGIR, SEKE, ARITH, FODO). Dr. Rishe has been awarded $15 million in research grants by Government and Industry. His research is currently sponsored by NASA ($5.5M), NSF ($4M), and other agencies. Dr. Rishe also has extensive experience in database applications and database systems in the industry. This included eight years of employment as head of software and database projects (1976-84) and later consulting for companies such as Hewlett-Packard and the telecommunications industry. Rishe is the founder and director of the High Performance Database Research Center at FIU, which now employs over 100 researchers. Rishe presently mentors 30 graduate students and 6 postdocs. Rishe has graduated 29 MS students and 7 PhD students.

Li Yang

Ms. Yang is currently a PhD candidate in School of Computer Science at Florida International University. She received her B.E. and M.E. in Computer Science and Engineering from Sichuan Union University in China in 1995 and 1998, respectively. Her research interests include XML, semi-structured data and

database systems, storage, indexing and query processing of XML, information integration, semantic web, data warehousing and data mining. She is a member of ACM, ACM SIGMOD and IEEE.

Maxim Chekmasov

Dr. Chekmasov is a visiting Assistant Professor at Florida International University (FIU), where he holds the position of General Manager of the High Performance Database Research Center. Dr. Chekmasov has a great deal of experience in database design and implementation, as well as research in database modeling. He received his PhD, MS, and BS degrees from St. Petersburgh University, Russia, in 1995, 1991, and 1989 respectively. Dr. Chekmasov has authored more than thirteen papers covering his areas of interest.

Scott C. Graham

Scott Graham is a Doctoral candidate at Florida International University (FIU), where he holds the position of Assistant Director for Government Relations at the High Performance Database Research Center. His areas of interest include database modeling, Internet applications, and computer architecture. Mr. Graham received his MS from FIU in 1992 and his BSE from the University of Florida in 1989. He has authored eighteen publications covering his areas of interest and is a member of both Tau Beta Pi and Upsilon Pi Epsilon honor societies.

Figures

Figure 1

DTD Running Example

```
<!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 address ANY>
            <!ELEMENT article (title, author*, contactauthor)>
            <!ELEMENT contactauthor EMPTY>
            <!ATTLIST contactauthor authorID IDREF   #IMPLIED>
] >
```

Figure 2

Publication Semantic Schema Example