# Formal Representation and Transformation of DTDs to Sem-ODM Semantic Schemas*

Li Yang
Department of Computer Science
University of West Georgia
Carrollton, GA 30118, U.S.A.

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

**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 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 Definition (DTD) and Semantic Schema of the Semantic Binary Object-Oriented Data Model (Sem-ODM) are formally defined and the conversion from DTD to Semantic Schema is presented. Through this formalization, we achieve preciseness and conciseness in expressing both data models and converting one to the other.*

**Keywords:** DTD, Sem-ODM Semantic Schema, Formal Definition, Schema Transformation

## 1. Introduction

With the growing popularity of XML (eXtensible Markup Language) [1] as a data exchange and representation format on the Web, many XML-related issues have been studied in different projects; for example [8, 4, 15,6, 9] focused on the storage issues of XML and [2, 7] 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 has formalized their work, even though formalization is a very important mechanism for understanding the data models and the transformation between them.

In this paper, we study the formalization aspect of the transformation between DTD (Document Type Definition) [1] and Sem-ODM (Semantic Binary Object-Oriented Data Model) [14]. We first present formal definitions of a DTD and a Semantic Schema. We then describe a formal framework in which a DTD is converted into a Semantic Schema. 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.

### 1.1 Related work

Several XML schema languages, such as DTD and XML Schema [5] among others, have been proposed to describe the structure and semantics of XML documents. [11] formally described several XML schema languages (DTD, XML Schema, RELAX [13], and others) based on regular tree languages. It represented a DTD as a local regular tree grammar, whereas [16] represented a DTD as an extended context free grammar and [3] presented a DTD in terms of Description Logic. Lee et al. [10] formalized relational schemas and DTDs, and presented a nesting-based translation algorithm to transform a relational schema into a DTD. In [12], 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 [10] and has been influenced by the formalism presented in [12]. However, [12] 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 DTD to a Sem-ODM Semantic Schema, not to a relational schema.

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

## 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. Section 4 concludes this paper and points out future work.

# 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 [12, 3, 10]. 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 $\epsilon$ 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 $\epsilon$ }. Note that u =$\epsilon$ represents no default value is provided.

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

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

In the above definition, $\epsilon$ 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.

```
<!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>

]>
```

**Fig. 1.** DTD Running Example

For example, the DTD in Figure 1 which is extracted from [15] and slightly modified can be represented formally as $G_1 = (E, A, S, P)$, where:

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

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

Note that we simplify the DTD before the mapping process so that the DTD does not contain the Choice content type, since none of relational or Sem-ODB databases supports that concept. 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. 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, such as person and book. The relations in a Semantic Schema are binary. Each of them is created from an abstract category (the *Domain* of the relation) to another category (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. 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}.

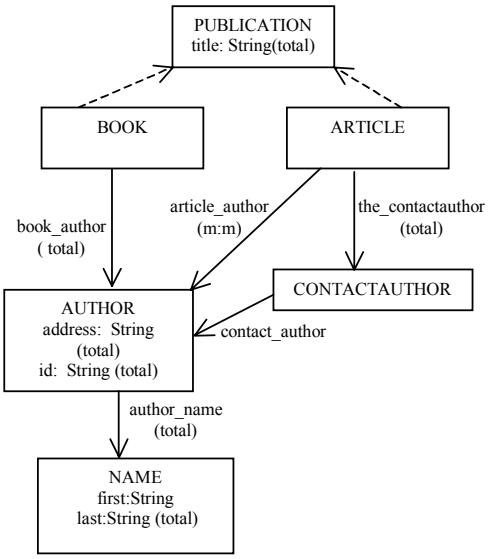


**Fig. 2.** Semantic Schema Example Representing Publication

[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 $xRy$.

**(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, $C_a \subseteq \widehat{C}_a$ ;
- $W$ 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, $C_c \subseteq \widehat{C}_c$ ;
- $R$ 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, ARTICLE, AUTHOR, CONTACTAUTHOR, NAME}
- $W$ ={(PUBLICATION, BOOK, ARTICLE)}
- $C_c$ ={String}
- $R$ ={title(m_1:total::publication → String), adress(m_1:total::AUTHOR → String), id(m_1: total::AUTHOR → String),
  first (m_1:not_total::NAME → String), last (m_1:total::NAME → String),
  book_author(m_1:total::BOOK → AUTHOR),
  article_author(m_m:not_total::ARTICLE → AUTHOR),
  the_contactauthor(m_1: total::ARTICLE → CONTACTAUTHOR),
  author_name(m_1: total::AUTHOR → NAME),
  contact_author(m_1:not_total::CONTACTAUTHOR → AUTHOR)}

# 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 mapping. The basic idea of our mapping algorithm 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, or 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 $Y \rightarrow (....X....)$ or $Y \rightarrow (....X?....)$, and no $Z \in E$ exists such that $Z \rightarrow r'$, $X$ appears in $r'$, 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 rules:
      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, *X* is mapped to a Category called *X* with one attribute *r* such that $r(c:t::X \rightarrow String)$, where *c*, and *t* are determined as follows:
   i. If $X \rightarrow$ (*PCDATA*), Then *c* = m_1, *t* = total
   ii. If $X \rightarrow$ (*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 $data(m\_1: total: title \rightarrow String)$. 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 $book\_title(m\_1: total::book \rightarrow title)$ and $article\_title(m\_1: total::article \rightarrow title)$ according to the rule 4 below.

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 $address(m\_1: total::author \rightarrow XMLType)$.
   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 way as in 3-b).

4. Relationship mapping
   a) Relationship in Sequence content.
   For every *X* in the form of $X \rightarrow (r_1, r_2, ....r_n)$, 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 $X \rightarrow (PCDATA \mid Y_1 \mid ... \mid .Y_n)*$, 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 $Y \rightarrow (....X....)$, then *c*, *t* and E are determined in the same procedure as in the following (2).
   b) If $Y \rightarrow (....X?....)$, 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 $\tau$ = CDATA, then
   i. If *d* = IMPLIED or $\epsilon$, then *c* =m_1, *t* =not_total, *E* =*String*
   ii. If *d* = REQUIRED, then *c* =m_1, *t* =total, *E* =String

b) If $\tau$ = ENUM, then
   i. If *d* = IMPLIED or $\epsilon$, 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 $\tau$ = ID, then
   i. If *d* = IMPLIED or $\epsilon$, 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 $\tau$ = IDREF, then
   i. If *d* = IMPLIED or $\epsilon$, 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., $authorID\ (m\_1: not\_total::contactauthor \rightarrow author)$.

e) If $\tau$ = IDREFS, then
   i. If *d* = IMPLIED or $\epsilon$, 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 $\tau$ = 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 }
- $W = \varnothing$
- $C_c$ ={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)}

Through the transformation, the structure and semantic information of a DTD are fully captured 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.

# 4. Conclusions 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 XML data model and Sem-ODM. 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 spe-

cial cases, while attributes and parent-child element relationships are converted into binary relations in the Sem-ODM. By formalizing the transformation, the differences, similarities, and possible connection of the two data models are expressed.

We are working on extending our work to XML Schema. Compared to DTD, XML Schema has significant enhancements. For instance, it supports more data types, inheritance, and type definition by extension and restriction, etc. Sem-ODM supports inheritance, and data types other than integer, string, etc. However, they are not equivalent in expressive power. To fully support XML Schema, extensions to Sem-ODM both semantically and syntactically have to be explored.

# 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] M. Carey, D. Florescu, et al., "XPERANTO: Publishing Object-Relational Data as XML", Proceedings of the 26th Int. Conf. On Very Large Data Bases (VLDB), Cairo, Egypt, 2000.

[3] D. Calvanese, G. D. Giacomo, M. 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] 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.

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

[6] 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.

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

[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] 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.

[10] D. Lee, M. Mani, F. Chiu, W. 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.

[11] M. Murata, D. Lee, M. Mani, "Taxonomy of XML Schema Languages using Formal Language Theory", Extreme Markup Languages, Montreal, Canada, August, 2000.

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

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

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

[15] 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.

[16] 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.