Schema Conversion Methods between XML and Relational Models

Dongwon Lee, Penn State University
Murali Mani and Wesley W. Chu, University of California, Los Angeles
dongwon@psu.edu, mani@cs.ucla.edu, wwc@cs.ucla.edu

Abstract. In this chapter, three semantics-based schema conversion methods are presented: 1) CPI converts an XML schema to a relational schema while preserving semantic constraints of the original XML schema, 2) NeT derives a nested structured XML schema from a flat relational schema by repeatedly applying the *nest* operator so that the resulting XML schema becomes hierarchical, and 3) CoT takes a relational schema as input, where multiple tables are interconnected through inclusion dependencies and generates an equivalent XML schema as output.

1 Introduction

Recently, XML [1] has emerged as the *de facto* standard for data formats on the web. The use of XML as the common format for representing, exchanging, storing, and accessing data poses many new challenges to database systems. Since the majority of everyday data is still stored and maintained in relational database systems, we expect that the needs to convert data formats between XML and relational models will grow substantially. To this end, several schema conversion algorithms have been proposed (e.g., [2, 3, 4, 5]). Although they work well for the given applications, the XML-to-Relational or Relational-to-XML conversion algorithms only capture the *structure* of the original schema and largely ignore the hidden *semantic constraints*. To clarify, consider the following DTD that models conference publications:

```
<!ELEMENT conf(title,soc,year,mon?,paper+)>
<!ELEMENT paper(pid,title,abstract?)>
```

Suppose the combination of `title` and `year` uniquely identifies the `conf`. Using the hybrid inlining algorithm [4], the DTD would be transformed to the following relational schema:

```
conf  (title,soc,year,mon)
paper (pid,title,conf_title,conf_year,abstract)
```

While the relational schema correctly captures the structural aspect of the DTD, it does not enforce correct semantics. For instance, it cannot prevent a tuple t_1: `paper(100,'DTD...',` `'ER',3000,'...')` from being inserted. However, tuple t_1 is inconsistent with the semantics of the given DTD since the DTD implies that the paper cannot exist without being associated with a conference and there is apparently no conference "ER-3000" yet. In

database terms, this kind of violation can be easily prevented by an *inclusion dependency* saying "`paper[conf_title,conf_year] ⊆ conf[title,year]`".

The reason for this inconsistency between the DTD and the transformed relational schema is that most of the proposed conversion algorithms, so far, have largely ignored the hidden *semantic constraints* of the original schema.

1.1 Related Work

Schema Conversion vs. Schema Matching: It is important to differentiate the problem that we deal with in this chapter, named as *schema conversion* problem, from another similar one known as *schema matching* problem. Given a *source* schema s_1 and a *target* schema t_1, the schema matching problem finds a "mapping" that relates elements in s_1 to ones in t_1. On the other hand, in the schema conversion problem, only a *source* schema s_2 is given and the goal is to find a *target* schema t_2 that is equivalent to s_2. Often, the source and target schemas in the schema matching problem belong to the same data model[1] (e.g., relational model), while they belong to different models in the schema conversion problem (e.g., relational and XML models). Schema matching problem itself is a difficult problem with many important applications and deserves special attention. For further discussion on the schema matching problem, refer to [6] (survey), [7] (latest development), etc.

Between XML and Non-relational Models: Schema conversion between different models has been extensively investigated. Historically, the trend for schema conversion has always been between consecutive models or models with overlapping time frames, as they have evolved (e.g., between Network and Relational models [8, 9], between ER and OO models [10, 11], or between UML and XML models [12, 13, 14, 15]). For instance, [16] deals with conversion problems in OODB area; since OODB is a richer environment than RDB, their work is not readily applicable to our application. The logical database design methods and their associated conversion techniques to other data models have been extensively studied in ER research. For instance, [17] presents an overview of such techniques. However, due to the differences between ER and XML models, those conversion techniques need to be modified substantially. In general, since works developed in this category are often ad hoc and were aimed at particular applications, it is not trivial to apply them to schema conversion between XML and relational models.

From XML to Relational: From XML to relational schema, several conversion algorithms have been proposed recently. STORED [2] is one of the first significant attempts to store XML data in relational databases. STORED uses a data mining technique to find a representative DTD whose support exceeds the pre-defined threshold and using the DTD, converts XML documents to relational format. Because [18] discusses template language-based conversion from DTD to relational schema, it requires human experts to write an XML-based conversion rule. [4] presents three inlining algorithms that focus on the table level of the schema conversions. On the contrary, [3] studies different performance issues among eight algorithms that focus on the attribute and value level of the schema. Unlike these, we propose a method where the hidden semantic constraints in DTDs are systematically found and translated into

---

[1]There are cases where schema matching problem deals with a mapping between different data models (e.g., [6]), but we believe most of such cases can be replaced by: 1) a schema conversion between different models, followed by 2) a schema matching within the same model.

relational formats [19]. Since the method is orthogonal to the structure-oriented conversion method, it can be used along with algorithms in [2, 18, 4, 3].

From Relational to XML: There have been different approaches for the conversion from the relational model to XML model, such as XML Extender from IBM, XML-DBMS, SilkRoute [20], and XPERANTO [5]. All of the above tools require the user to specify the mapping from the given relational schema to XML schema. In XML Extender, the user specifies the mapping through a language such as DAD or XML Extender Transform Language. In XML-DBMS, a template-driven mapping language is provided to specify the mappings. SilkRoute provides a declarative query language (RXL) for viewing relational data in XML. XPERANTO uses XML query language for viewing relational data in XML. Note that in SilkRoute and XPERANTO, the user has to specify the query in the appropriate query language.

1.2 Overview of Three Schema Translation Algorithms

In this chapter, we present three schema conversion algorithms that not only capture the structure, but also the semantics of the original schema.

1. **CPI** (Constraints-preserving Inlining Algorithm): identifies various semantics constraints in the original XML schema and preserves them by rewriting them in the final relational schema.

2. **NeT** (Nesting-based Translation Algorithm): derives a nested structure from a flat relational schema by repeatedly applying the *nest* operator so that the resulting XML schema becomes hierarchical. The main idea is to find a more intuitive element content model of the XML schema that utilizes the regular expression operators provided by the XML schema specification (e.g., "*" or "+").

3. **CoT** (Constraints-based Translation Algorithm): Although NeT infers hidden characteristics of data by nesting, it is only applicable to a single table at a time. Therefore, it is unable to capture the overall picture of relational schema where multiple tables are interconnected. To remedy this problem, CoT considers inclusion dependencies during the translation, and merges multiple inter-connected tables into a coherent and hierarchical parent-child structure in the final XML schema.

2 The CPI Algorithm

Transforming a hierarchical XML model to a flat relational model is not a trivial task due to several inherent difficulties such as non-trivial 1-to-1 mapping, existence of set values, complicated recursion, and/or fragmentation issues [4]. Most XML-to-Relational conversion algorithms (e.g., [18, 2, 3, 4]) have so far focused mainly on the issue of structural conversion, largely ignoring the semantics that already existed in the original XML schema. Let us first describe various semantic constraints that one can mine from the DTD. Throughout the discussion, we will use the DTD and XML document in Tables 1 and 2 as examples.

```
<!ELEMENT conf     (title,date,editor?,paper*)>
<!ATTLIST conf     id      ID          #REQUIRED>
<!ELEMENT title    (#PCDATA)>
<!ELEMENT date     EMPTY>
<!ATTLIST date     year    CDATA       #REQUIRED
                   mon     CDATA       #REQUIRED
                   day     CDATA       #IMPLIED>
<!ELEMENT editor   (person*)>
<!ATTLIST editor   eids    IDREFS      #IMPLIED>
<!ELEMENT paper    (title,contact?,author,cite?)>
<!ATTLIST paper    id      ID          #REQUIRED>
<!ELEMENT contact EMPTY>
<!ATTLIST contact aid      IDREF       #REQUIRED>
<!ELEMENT author   (person+)>
<!ATTLIST author   id      ID          #REQUIRED>
<!ELEMENT person   (name,(email|phone)?)>
<!ATTLIST person   id      ID          #REQUIRED>
<!ELEMENT name     EMPTY>
<!ATTLIST name     fn      CDATA       #IMPLIED
                   ln      CDATA       #REQUIRED>
<!ELEMENT email    (#PCDATA)>
<!ELEMENT phone    (#PCDATA)>
<!ELEMENT cite     (paper*)>
<!ATTLIST cite     id      ID          #REQUIRED
                   format (ACM|IEEE)  #IMPLIED>
```

Table 1: A DTD for `Conference`.

2.1 Semantic Constraints in DTDs

Cardinality Constraints: In a DTD declaration, there are only 4 possible cardinality relationships between an element and its sub-elements as illustrated below:

```
<!ELEMENT article (title, author+, ref*, price?)>
```

1. (0,1): An element can have either zero or one sub-element. (e.g., sub-element `price`)

2. (1,1): An element must have one and only one sub-element. (e.g., sub-element `title`)

3. (0,N): An element can have zero or more sub-elements. (e.g., sub-element `ref`)

4. (1,N): An element can have one or more sub-elements. (e.g., sub-element `author`)

Following the notations in [17], let us call each cardinality relationship as type (0,1), (1,1), (0,N), (1,N), respectively. From these cardinality relationships, three major constraints can be inferred. The first is whether or not the sub-element can be null. We use the notation "$X \nrightarrow \emptyset$" to denote that an element X cannot be null. This constraint is easily enforced by the `NULL` or `NOT NULL` clause in SQL. The second is whether or not more than one sub-element can occur. This is also known as *singleton constraint* in [21] and is one kind of equality-generating dependencies. The third, given an element, whether or not its sub-element should occur. This is one kind of tuple-generating dependencies. The second and third types will be further discussed below.

Inclusion Dependencies (INDs): An *Inclusion Dependency* assures that values in the columns of one fragment must also appear as values in the columns of other fragments and is a generalization of the notion of *referential integrity*.

A trivial form of INDs found in the DTD is that "given an element X and its sub-element Y, Y must be included in X (i.e., $Y \subseteq X$)". For instance, from the `conf` element and its

```
<conf id="er05">
  <title>Int'l Conf. on Conceptual Modeling</title>
  <date> <year>2005</year> <mon>May</mon> <day>20</day> </date>
  <editor eids="sheth bossy">
    <person id="klavans">
      <name fn="Judith" ln="Klavans"/>  <email>klavans@cs.columbia.edu</email>
    </person>
  </editor>
  <paper id="p1">
    <title>Indexing Model for Structured...</title>
    <contact aid="dao"/>
    <author>  <person id="dao"><name fn="Tuong" ln="Dao"/>  </author>
  </paper>
  <paper id="p2">
    <title>Logical Information Modeling...</title>
    <contact aid="shah"/>
    <author>
      <person id="shah">  <name fn="Kshitij" ln="Shah"/> </person>
      <person id="sheth">
        <name fn="Amit" ln="Sheth"/> <email>amit@cs.uga.edu</email>
      </person>
    </author>
    <cite id="c100" format="ACM">
      <paper id="p3">
        <title>Making Sense of Scientific...</title>
        <author>
          <person id="bossy">
            <name fn="Marcia" ln="Bossy"/> <phone>391.4337</phone>
          </person>
        </author> </paper> </cite> </paper>
</conf>
<paper id="p7">
  <title>Constraints-preserving Trans...</title>
  <contact aid="lee"/>
  <author>
    <person id="lee">
      <name fn="Dongwon" ln="Lee"/>  <email>dongwon@cs.ucla.edu</email>
    </person> </author>
  <cite id="c200" format="IEEE"/>
</paper>
```

Table 2: An example XML document conforming to the DTD in Table 1.

four sub-elements in the `Conference` DTD, the following INDs can be found as long as `conf` is not null: {`conf.title` ⊆ `conf`, `conf.date` ⊆ `conf`, `conf.editor` ⊆ `conf`, `conf.paper` ⊆ `conf`}. Another form of INDs can be found in the attribute definition part of the DTD with the use of the `IDREF(S)` keyword. For instance, consider the `contact` and `editor` elements in the `Conference` DTD shown below:

```
<!ELEMENT person  (name,(email|phone)?>
<!ATTLIST person  id   ID      #REQUIRED>
<!ELEMENT contact EMPTY>
<!ATTLIST contact aid  IDREF   #REQUIRED>
<!ELEMENT editor  (person*)>
<!ATTLIST editor  eids IDREFS #IMPLIED>
```

The DTD restricts the `aid` attribute of the `contact` element such that it can only point to the `id` attribute of the `person` element[2]. Further, the `eids` attribute can only point to multiple `id` attributes of the `person` element. As a result, the following INDs can be derived: {`editor.eids` ⊆ `person.id`, `contact.aid` ⊆ `person.id`}. Such INDs can best be enforced by the "foreign key" if the attribute being referenced is a primary key. Otherwise, it needs to use the `CHECK`, `ASSERTION`, or `TRIGGERS` facility of SQL.

Equality-Generating Dependencies (EGDs): The *Singleton Constraint* [21] restricts an element to have "at most" one sub-element. When an element type X satisfies the singleton constraint towards its sub-element type Y, if an element instance x of type X has *two* sub-elements instances y_1 and y_2 of type Y, then y_1 and y_2 must be the same. This

---

[2]Precisely, an attribute with `IDREF` type does not specify which element it should point to. This information is available only by human experts. However, new XML schema languages such as XML-Schema and DSD can express where the reference actually points to [22].

Relationship	Symbol	not null	EGDs	TGDs
(0,1)	?	no	yes	no
(1,1)		yes	yes	yes
(0,N)	*	no	no	no
(1,N)	+	yes	no	yes

Table 3: Cardinality relationships and their corresponding semantic constraints.

property is known as *Equality-Generating Dependencies (EGDs)* and is denoted by "$X \rightarrow Y$" in database theory. For instance, two EGDs: $\{$conf $\rightarrow$ conf.title, conf $\rightarrow$ conf.date$\}$ can be derived from the conf element in Table 1. This kind of EGDs can be enforced by an SQL UNIQUE construct. In general, EGDs occur in the case of the (0,1) and (1,1) mappings in the cardinality constraints.

Tuple-Generating Dependencies (TGDs): TGDs in a relational model require that some tuples of a certain form be present in the table and use the "$\twoheadrightarrow$" symbol. Two useful forms of TGDs from DTD are the *child* and *parent constraints* [21].

1. **Child constraint:** "Parent $\twoheadrightarrow$ Child" states that every element of type $Parent$ must have at least one child element of type $Child$. This is the case of the (1,1) and (1,N) mappings in the cardinality constraints. For instance, from the DTD in Table 1, because the conf element must contain the title and date sub-elements, the child constraint conf $\twoheadrightarrow \{$title, date$\}$ holds.

2. **Parent constraint:** "Child $\twoheadrightarrow$ Parent" states that every element of type $Child$ must have a parent element of type $Parent$. According to XML specification, XML documents can start from any level of element without necessarily specifying its parent element, when a root element is not specified by <!DOCTYPE root>. In the DTD in Table 1, for instance, the editor and date elements can have the conf element as their parent. Further, if we know that all XML documents were started at the conf element level, rather than the editor or date level, then the parent constraint $\{$editor, date$\} \twoheadrightarrow$ conf holds. Note that the title $\twoheadrightarrow$ conf does not hold since the title element can be a sub-element of either the conf or paper element.

2.2   *Discovering and Preserving Semantic Constraints from DTDs*

The CPI algorithm utilizes a structure-based conversion algorithm as a basis and identifies various semantic constraints described in Section 2.1. We will use the *hybrid* algorithm [4] as the basis algorithm. CPI first constructs a *DTD graph* that represents the structure of a given DTD. A DTD graph can be constructed when parsing the given DTD. Its nodes are elements, attributes, or operators in the DTD. Each element appears exactly once in the graph, while attributes and operators appear as many times as they appear in the DTD. CPI then annotates various cardinality relationships (summarized in Table 3) among nodes to each edge of the DTD graph. Note that the cardinality relationship types in the graph consider not only element vs. sub-element relationships but also element vs. attribute relationships. Figure 1 illustrates an example of such an annotated DTD graph for the Conference DTD in Table 1.
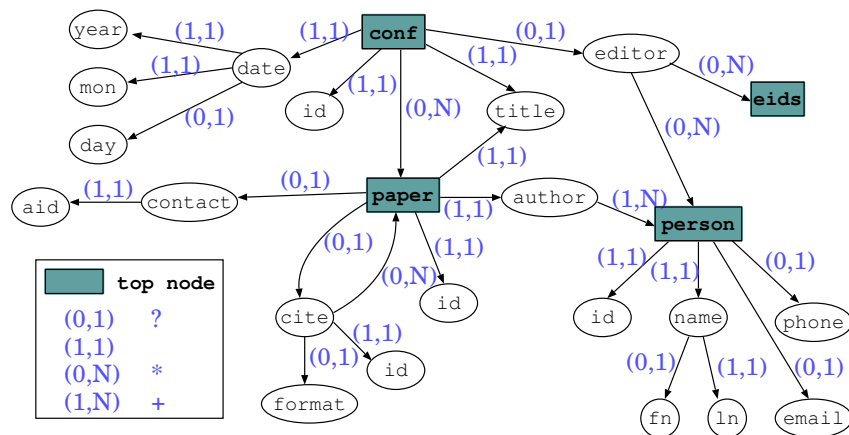
Figure 1: An annotated *DTD graph* for the `Conference` DTD in Table 1.

```
CREATE TABLE paper (
  id          NUMBER      NOT NULL,
  title       VARCHAR(50) NOT NULL,
  contact_aid VARCHAR(20),
  cite_id     VARCHAR(20),
  cite_format VARCHAR(50) CHECK (VALUE IN ("ACM", "IEEE")),
  root_elm    VARCHAR(20) NOT NULL,
  parent_elm  VARCHAR(20),
  fk_cite     VARCHAR(20) CHECK (fk_cite IN (SELECT cite_id FROM paper)),
  fk_conf     VARCHAR(20),
  PRIMARY KEY (id),
  UNIQUE (cite_id),
  FOREIGN KEY (fk_conf) REFERENCES conf(id),
  FOREIGN KEY (contact_aid) REFERENCES person(id)
);
```

Figure 2: Final relational "schema" for the `paper` element in the `Conference` DTD in Table 1, generated by CPI algorithm.

Once the annotated DTD graph is constructed, CPI follows the basic navigation method provided by the *hybrid* algorithm; it identifies **top nodes** [4, 19] that are the nodes: 1) not reachable from any nodes (e.g., source node), 2) direct child of "*" or "+" operator node, 3) recursive node with indegree > 1, or 4) one node between two mutually recursive nodes with indegree $= 1$. Then, starting from each top node T, *inline* all the elements and attributes at *leaf nodes* reachable from T unless they are other top nodes. In doing so, each annotated cardinality relationship can be properly converted to its counterpart in SQL syntax as described in Section 2.1. The details of the algorithm are beyond the scope of this chapter and interested readers can refer to [19]. For instance, Figure 2 and Table 4 are such output relational schema and data in SQL notation, automatically generated by the CPI algorithm.

3 The NeT Algorithm

The simplest Relational-to-XML translation method, termed as FT (Flat Translation) in [23], is to translate 1) tables in a relational schema to elements in an XML schema and 2) columns in a relational schema to attributes in an XML schema. FT is a simple and effective translation

paper								
id	root_elm	parent_elm	fk_conf	fk_cite	title	contact_aid	cite_id	cite_format
p1	conf	conf	er05	–	Indexing ...	dao	–	–
p2	conf	conf	er05	–	Logical ...	shah	c100	ACM
p3	conf	cite	–	c100	Making ...	–	–	–
p7	paper	–	–	–	Constraints ...	lee	c200	IEEE

Table 4: Final relational "data" for the `paper` element in the `Conference` DTD in Table 1, generated by CPI algorithm.

algorithm. However, since FT translates the "flat" relational model to a "flat" XML model in a one-to-one manner, it does not utilize several basic "non-flat" features provided by the XML model for data modeling, such as representing *repeating sub-elements* through regular expression operators (e.g., "*", "+"). To remedy the shortcomings of FT, we propose the NeT algorithm that utilizes various *element content models* of the XML model. NeT uses the *nest* operator [24] to derive a "good" element content model.

Informally, for a table t with a set of columns C, *nesting* on a non-empty column $X \in C$ collects all tuples that agree on the remaining columns $C - X$ into a set[3]. Formally,

Definition 1 (Nest). *[24]. Let t be a n-ary table with column set C, and $X \in C$ and $\overline{X} = C - X$. For each $(n - 1)$-tuple $\gamma \in \Pi_{\overline{X}}(t)$, we define an n-tuple γ^* as follows: $\gamma^*[\overline{X}] = \gamma$, and $\gamma^*[X] = \{\kappa[X] \mid \kappa \in t \wedge \kappa[\overline{X}] = \gamma\}$. Then, $nest_X(t) = \{\gamma^* \mid \gamma \in \Pi_{\overline{X}}(t)\}$.*

After $nest_X(t)$, if column X only has a set with "single" value $\{v\}$ for all the tuples, then we say that **nesting failed** and we treat $\{v\}$ and v interchangeably (i.e., $\{v\} = v$). Thus when nesting failed, the following is true: $nest_X(t) = t$. Otherwise, if column X has a set with "multiple" values $\{v_1, ..., v_k\}$ with $k \geq 2$ for at least one tuple, then we say that **nesting succeeded**.

Example 1. *Consider a table R in Table 5. Here we assume that the columns A, B, and C are non-nullable. In computing $nest_A(R)$ at (b), the first, third, and fourth tuples of R agree on their values in columns (B, C) as (a, 10), while their values of the column A are all different. Therefore, these different values are grouped (i.e., nested) into a set $\{1,2,3\}$. The result is the first tuple of the table $nest_A(R) - (\{1,2,3\}, a, 10)$. Similarly, since the sixth and seventh tuples of R agree on their values as (b, 20), they are grouped to a set $\{4,5\}$. In computing $nest_B(R)$ at (c), there are no tuples in R that agree on the values of the columns (A, C). Therefore, $nest_B(R) = R$. In computing $nest_C(R)$ at (d), since the first two tuples of $R - (1, a, 10)$ and (1, a, 20) – agree on the values of the columns (A, B), they are grouped to (1, a, $\{10,20\}$). Nested tables (e) through (j) are constructed similarly.*

Since the *nest* operator requires scanning of the entire set of tuples in a given table, it can be quite expensive. In addition, as shown in Example 1, there are various ways to nest the given table. Therefore, it is important to find an efficient way (that uses the *nest* operator a minimum number of times) of obtaining an acceptable element content model. For a detailed description on the various properties of the *nest* operator, the interested are referred to [23, 25].

Lemma 1. *Consider a table t with column set C, candidate keys, $K_1, K_2, \ldots, K_n \subseteq C$, and column set K such that $K = K_1 \cap K_2 \cap \ldots \cap K_n$. Further, let $|C| = n$ and $|K| = m$ ($n \geq m$). Then, the number of necessary nestings, N, is bounded by $N \leq \sum_{k=1}^{m} m^{\underline{k}}$*

---

[3]Here, we only consider single attribute nesting.

	A	B	C
#1	1	a	10
#2	1	a	20
#3	2	a	10
#4	3	a	10
#5	4	b	10
#6	4	b	20
#7	5	b	20

(a) R

A^+	B	C
{1,2,3}	a	10
1	a	20
4	b	10
{4,5}	b	20

(b) $nest_A(R)$

A	B	C
1	a	10
1	a	20
2	a	10
3	a	10
4	b	10
4	b	20
5	b	20

(c) $nest_B(R) = R$

A	B	C^+
1	a	{10,20}
2	a	10
3	a	10
4	b	{10,20}
5	b	20

(d) $nest_C(R)$

A^+	B	C
{1,2,3}	a	10
1	a	20
4	b	10
{4,5}	b	20

(e) $nest_B(nest_A(R))$ $= nest_C(nest_A(R))$

A^+	B	C^+
1	a	{10,20}
{2,3}	a	10
4	b	{10,20}
5	b	20

(f) $nest_A(nest_C(R))$

A	B	C^+
1	a	{10,20}
2	a	10
3	a	10
4	b	{10,20}
5	b	20

(g) $nest_B(nest_C(R))$

A^+	B	C
{1,2,3}	a	10
1	a	20
4	b	10
{4,5}	b	20

(h) $nest_C(nest_B(nest_A(R)))$ $= nest_B(nest_C(nest_A(R)))$

A^+	B	C^+
1	a	{10,20}
{2,3}	a	10
4	b	{10,20}
5	b	20

(i) $nest_B(nest_A(nest_C(R)))$ $= nest_A(nest_B(nest_C(R)))$

Table 5: A relational table R and its various nested forms. Column names containing a set after nesting (i.e., nesting succeeded) are appended by "+" symbol.

Lemma 1 implies that when candidate key information is available, one can avoid unnecessary nestings substantially. For instance, suppose attributes A and C in Table 5 constitute a key for R. Then, one needs to compute only: $nest_A(R)$ at (b), $nest_C(R)$ at (d), $nest_C(nest_A(R))$ at (e), $nest_A(nest_C(R))$ at (f) in Table 5.

After applying the $nest$ operator to the given table repeatedly, there may remain several nested tables where nesting succeeded. In general, the choice of the final schema should take into consideration the semantics and usages of the underlying data or application and this is where user intervention is beneficial. By default, without further input from users, NeT chooses the nested table where the most number of nestings succeeded as the final schema, since this is a schema which provides low "data redundancy". The outline of the NeT algorithm is as follows:

1. For each table t_i in the input relational schema $\mathbb{R}$, apply the $nest$ operator repeatedly until no nesting succeeds.

2. Choose the best nested table based on the selected criteria. Denote this table as $t'_i(c_1, \ldots, c_{k-1}, c_k, \ldots, c_n)$, where nesting succeeded on the columns $\{c_1, \ldots, c_{k-1}\}$.

   (a) If $k = 1$, follow the FT translation.

   (b) If $k > 1$,

      i. For each column c_i ($1 \leq i \leq k - 1$), if c_i was nullable in $\mathbb{R}$, use c_i^* for the element content model, and c_i^+ otherwise.

ii. For each column c_j ($k \leq j \leq n$), if c_i was nullable in $\mathbb{R}$, use $c_j^?$ for the element content model, and c_j otherwise.

4   The CoT Algorithm

The NeT algorithm is useful for decreasing data redundancy and obtaining a more intuitive schema by 1) removing redundancies caused by multivalued dependencies, and 2) performing grouping on attributes. However, NeT considers tables one at a time, and cannot obtain an *overall picture* of the relational schema where many tables are interconnected with each other through various other dependencies. To remedy this problem, we propose the CoT algorithm that uses Inclusion Dependencies (INDs) of relational schema. General forms of INDs are difficult to acquire from the database automatically. However, we shall consider the most pervasive form of INDs, foreign key constraints, which can be queried through ODBC/JDBC interface.

The basic idea of the CoT is the following: For two distinct tables s and t with lists of columns X and Y, respectively, suppose we have a foreign key constraint $s[\alpha] \subseteq t[\beta]$, where $\alpha \subseteq X$ and $\beta \subseteq Y$. Also suppose that $K_s \subseteq X$ is the key for s. Then, different cardinality binary relationships between s and t can be expressed in the relational model by a combination of the following: 1) α is unique/not-unique, and 2) α is nullable/non-nullable. Then, the translation of two tables s, t with a foreign key constraint works as follows:

1. If α is non-nullable (i.e., none of the columns of α can take null values), then:

   (a) If α is unique, then there is a $1 : 1$ relationship between s and t, and can be captured as
      `<!ELEMENT t (Y, s?)>`.

   (b) If α is not-unique, then there is a $1 : n$ relationship between s and t, and can be captured as
      `<!ELEMENT t (Y, s*)>`.

2. If s is represented as a sub-element of t, then the key for s will change from K_s to $(K_s - \alpha)$. The key for t will remain the same.

Extending this to the general case where multiple tables are interconnected via INDs, consider the schema with a set of tables $\{t_1, ..., t_n\}$ and INDs $t_i[\alpha_i] \subseteq t_j[\beta_j]$, where $i, j \leq n$. We consider only those INDs that are foreign key constraints (i.e., β_j constitutes the primary key of the table t_j), and where α_i is non-nullable. The relationships among tables can be captured by a graphical representation, termed IND-Graph.

Definition 2 (IND-Graph). *An* IND-Graph $G = (V, E)$ *consists of a node set V and a directed edge set E, such that for each table t_i, there exists a node $V_i \in V$, and for each distinct IND $t_i[\alpha] \subseteq t_j[\beta]$, there exists an edge $E_{ji} \in E$ from the node V_j to V_i.*

Note the edge direction is reversed from the IND direction for convenience. Given a set of INDs, the IND-Graph can be easily constructed. Once an IND-Graph G is constructed, CoT needs to decide the starting point to apply translation rules. For that purpose, we use the notion of **top nodes**. Intuitively, an element is a top node if it *cannot* be represented as a sub-element of any other element. Let T denote the set of top nodes. Then, CoT traverses G, using a Breadth-First Search (BFS), until it traverses all the nodes and edges, while capturing the INDs on edges as either sub-elements (when the node is visited for the first time) or IDREF attributes (when the node was visited already).

```
student(Sid, Name, Advisor)
emp(Eid, Name, ProjName)
prof(Eid, Name, Teach)
course(Cid, Title, Room)
dept(Dno, Mgr)
proj(Pname, Pmgr)
```

```
student(Advisor) ⊆ prof(Eid)
emp(ProjName) ⊆ proj(Pname)
prof(Teach) ⊆ course(Cid)
prof(Eid, Name) ⊆ emp(Eid, Name)
dept(Mgr) ⊆ emp(Eid)
proj(Pmgr) ⊆ emp(Eid)
```

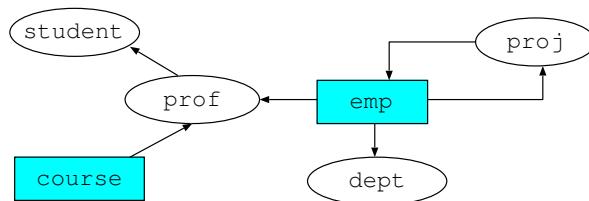Table 6: An example schema with associated INDs.

Figure 3: The IND-Graph representation of the schema in Table 6 (*top nodes* denoted by rectangular nodes).

Example 2. *Consider a schema and its associated INDs in Table 6. The IND-Graph with two top nodes is shown in Figure 3: 1)* `course`*: There is no node* t*, where there is an IND of the form* `course`$[\alpha] \subseteq t[\beta]$*, and 2)* `emp`*: There is a cyclic set of INDs between* `emp` *and* `proj`*, and there exists no node* t *such that there is an IND of the form* `emp`$[\alpha] \subseteq t[\beta]$ *or* `proj`$[\alpha] \subseteq t[\beta]$*. Then,*

- *First, starting from a top node* `course`*, do a BFS scan. Pull up a reachable node* `prof` *into* `course` *and label it as a sub-element by* `<!ELEMENT course (Cid, Title, Room, prof*)>`*. Similarly, the node* `student` *is also pulled up into its parent node* `prof` *by* `<!ELEMENT prof (Eid, Name, student*)>`*. Since the node* `student` *is a leaf, no nodes can be pulled in:* `<!ELEMENT student (Sid, Name)>`*. Since there is no more unvisited reachable node from* `course`*, the scan stops.*

- *Next, starting from another top node* `emp`*, pull up neighboring node* `dept` *into* `emp` *similarly by* `<!ELEMENT emp (Eid, Name, ProjName, dept*)>` *and* `<!ELEMENT dept (Dno, Mgr)>`*. Then, visit a neighboring node* `prof`*, but* `prof` *was visited already. To avoid data redundancy, an attribute* `Ref_prof` *is added to* `emp` *accordingly. Since attributes in the left-hand side of the corresponding IND,* $prof(Eid, Name) \subseteq emp(Eid, Name)$*, form a super key, the attribute* `Ref_prof` *is assigned type* `IDREF`*, and not* `IDREFS`*:* `<!ATTLIST prof Eid ID>` *and* `<!ATTLIST emp Ref_prof IDREF>`*.*

- *Next, visit a node* `proj` *and pull it up to* `emp` *by* `<!ELEMENT emp (Eid, Name, ProjName, dept*, proj*)>` *and* `<!ELEMENT proj (Pname)>`*. In the next step, visit a node* `emp` *from* `prof`*. Since it was already visited, an attribute* `Ref_emp` *of type* `IDREFS` *is added to* `proj`*, and the scan stops.*

It is worthwhile to point out that there are several places in CoT where human experts can help to find a better mapping based on the semantics and usages of the underlying data or application.

DTD Semantics		DTD Schema		Relational Schema			
Name	Domain	Elm/Attr	ID/IDREF(S)	Table/Attr	$\rightarrow$	$\twoheadrightarrow$	$\twoheadrightarrow \emptyset$
`novel`	literature	10/1	1/0	5/13	6	9	9
`play`	Shakespeare	21/0	0/0	14/46	17	30	30
`tstmt`	religious text	28/0	0/0	17/52	17	22	22
`vCard`	business card	23/1	0/0	8/19	18	13	13
`ICE`	content synd.	47/157	0/0	27/283	43	60	60
`MusicML`	music desc.	12/17	0/0	8/34	9	12	12
`OSD`	s/w desc.	16/15	0/0	15/37	2	2	2
`PML`	web portal	46/293	0/0	41/355	29	36	36
`Xbel`	bookmark	9/13	3/1	9/36	9	1	1
`XMI`	metadata	94/633	31/102	129/3013	10	7	7
`BSML`	DNA seq.	112/2495	84/97	104/2685	99	33	33

Table 7: Summary of CPI algorithm.

5 Experimental Results

5.1 CPI Results

CPI was tested against DTDs gathered from OASIS[4]. For all cases, CPI successfully identified hidden semantic constraints from DTDs and correctly preserved them by rewriting them in SQL. Table 7 shows a summary of our experimentation. Note that people seldom used the `ID` and `IDREF(S)` constructs in their DTDs except in the `XMI` and `BSML` cases. The number of tables generated in the relational schema was usually smaller than that of elements/attributes in DTDs due to the inlining effect. The only exception to this phenomenon was the `XMI` case, where extensive use of types (0,N) and (1,N) cardinality relationships resulted in many top nodes in the ADG.

The number of semantic constraints had a close relationship with the design of the DTD hierarchy and the type of cardinality relationship used in the DTD. For instance, the `XMI` DTD had many type (0,N) cardinality relationships, which do not contribute to the semantic constraints. As a result, the number of semantic constraints at the end was small, compared to that of elements/attributes in the DTD. This was also true for the `OSD` case. On the other hand, in the `ICE` case, since it used many type (1,1) cardinality relationships, it resulted in many semantic constraints.

5.2 NeT Results

Our preliminary results comparing the goodness of the *XSchema* obtained from NeT and FT with that obtained from DB2XML v 1.3 [26] appeared in [23]. We further applied our NeT algorithm on several test sets drawn from UCI KDD[5] / ML[6] repositories, which contain a multitude of single-table relational schemas and data. Sample results are shown in Table 8. Two metrics are shown in Figure 4(a). A high value for *NestRatio* shows that we did not perform unnecessary nesting and the high value for *ValueRatio* shows that the nesting removed a great deal of redundancy.

---

[4]http://www.oasis-open.org/cover/xml.html

[5]http://kdd.ics.uci.edu/

[6]http://www.ics.uci.edu/~mlearn/MLRepository.html

Test Set	Attr. / tuple	NestRatio	ValueRatio	Size before / after	Nested attr.	Time (sec.)
Balloons1	5 / 16	42 / 64	80 / 22	0.455 / 0.152	3	1.08
Balloons2	5 / 16	42 / 64	80 / 22	0.455 / 0.150	3	1.07
Balloons3	5 / 16	40 / 64	80 / 42	0.455 / 0.260	3	1.14
Balloons4	5 / 16	42 / 64	80 / 22	0.455 / 0.149	3	1.07
Hayes	6 / 132	1 / 6	792 / 522	1.758 / 1.219	1	1.01
Bupa	7 / 345	0 / 7	2387 / 2387	7.234 / 7.234	0	4.40
Balance	5 / 625	56 / 65	3125 / 1120	6.265 / 2.259	4	21.48
TA_Eval	6 / 110	253 / 326	660 / 534	1.559 / 1.281	5	24.83
Car	7 / 1728	1870 / 1957	12096 / 779	51.867 / 3.157	6	469.47
Flare	13 / 365	11651 / 13345	4745 / 2834	9.533 / 5.715	4	6693.41

Table 8: Summary of NeT experimentations.

$$\text{NestRatio} = \frac{\text{\# of successful nesting}}{\text{\# of total nesting}}$$

$$\text{ValueRatio} = \frac{\text{\# of original data values}}{\text{\# of data values in nested table}}$$

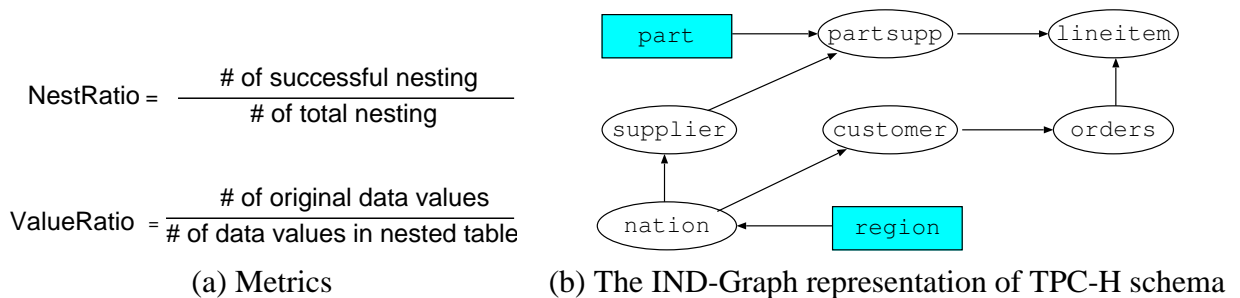(a) Metrics   (b) The IND-Graph representation of TPC-H schema

Figure 4: Metrics and IND-Graph.

In our experimentation[7], we observed that most of the attempted nestings were successful, and hence our optimization rules are quite efficient. In Table 8, we see that nesting was useful for all data sets except for the `Bupa` data set. Also nesting was *especially* useful for the `Car` data set, where the size of the nested table is only 6% of the original data set. Time required for nesting is an important parameter, and it jointly depends on the number of attempted nestings and the number of tuples. The number of attempted nestings depends on the number of attributes, and increases drastically as the number of attributes increases. This is observed for the `Flare` data set, where we have to do nesting on 13 attributes.

5.3 CoT Results

For testing CoT, we need some well-designed relational schema where tables are interconnected via inclusion dependencies. For this purpose, we use the TPC-H schema v 1.3.0[8], which is an ad-hoc, decision support benchmark and has 8 tables and 8 inclusion dependencies. The IND-Graph for the TPC-H schema is shown in Figure 4(b). CoT identified two top-nodes – `part` and `region`, and eventually generated the XML document having interwoven hierarchical structures; six of the eight inclusion dependencies are mapped using a sub-element, and the remaining two are mapped using `IDREF` attributes.

Figure 5 shows a comparison of the number of data values originally present in the database, and the number of data values in the XML document generated by FT and CoT. Because FT is a flat translation, the number of data values in the XML document generated

---

[7]Available at http://www.cs.ucla.edu/~mani/xml

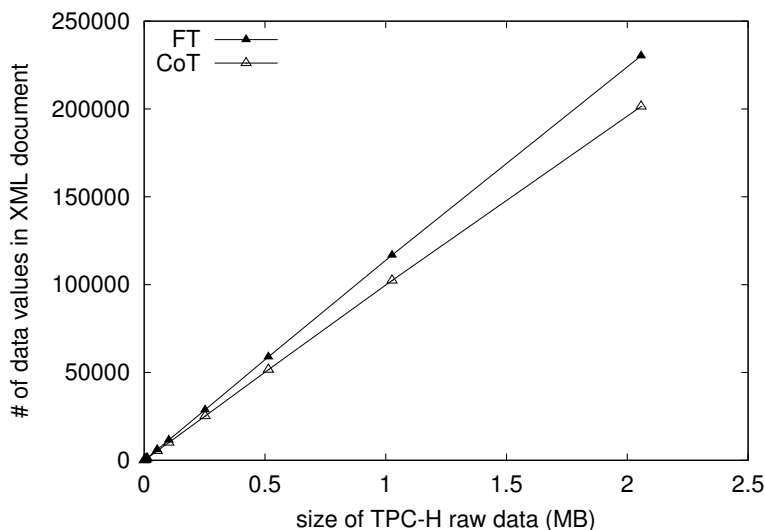[8]http://www.tpc.org/tpch/spec/h130.pdf

Figure 5: Size comparison of two algorithms.

by FT is the same as the number of data values in the original data. However, CoT is able to decrease the number of data values in the generated XML document by more than 12%.

6 Conclusion

We have presented a method to transform a relational schema to an XML schema, and two methods to transform an XML schema to a relational schema, both in *structural* and *semantic* aspects. All three algorithms are "correct" in the sense that they have all preserved the original information of relational schema. For instance, using the notion of information capacity [27], a theoretical analysis for the correctness of our translation procedures is possible; we can actually show that CPI, NeT and CoT algorithms are *equivalence preserving conversions*.

Despite the difficulties in conversions between XML and relational models, there are many practical benefits. We strongly believe that devising more accurate and efficient conversion methodologies between XML and relational models is important. The prototypes of our algorithms are available at: http://www.cobase.cs.ucla.edu/projects/xpress/

References

[1] Bray, T., Paoli, J., Sperberg-McQueen (Eds), C.M.: "Extensible Markup Language (XML) 1.0 (2nd Edition)". W3C Recommendation (2000) http://www.w3.org/TR/2000/REC-xml-20001006.

[2] Deutsch, A., Fernandez, M.F., Suciu, D.: "Storing Semistructured Data with STORED". In: ACM SIGMOD International Conference on Management of Data, Philadephia, PA (1998)

[3] Florescu, D., Kossmann, D.: "Storing and Querying XML Data Using an RDBMS". IEEE Data Engineering Bulletin **22** (1999) 27–34

[4] Shanmugasundaram, J., Tufte, K., He, G., Zhang, C., DeWitt, D., Naughton, J.: "Relational Databases for Querying XML Documents: Limitations and Opportunities". In: International Conference on Very Large Data Bases, Edinburgh, Scotland (1999)

[5] Carey, M., Florescu, D., Ives, Z., Lu, Y., Shanmugasundaram, J., Shekita, E., Subramanian, S.: "XPERANTO: Publishing Object-Relational Data as XML". In: International Workshop on the Web and Databases, Dallas, TX (2000)

[6] Madhavan, J., Berstein, P.A., Rahm, E.: "Generic Schema Matching with Cupid". In: International Conference on Very Large Data Bases, Roma, Italy (2001)

[7] Miller, R.J., Haas, L., Hernandez, M.A.: "Schema Mapping as Query Discovery". In: International Conference on Very Large Data Bases, Cairo, Egypt (2000)

[8] Navathe, S.B.: "An Intuitive Approach to Normalize Network Structured Data". In: International Conference on Very Large Data Bases, Montreal, Quebec, Canada (1980)

[9] Lien, Y.E.: "On the Equivalence of Database Models". Journal of the ACM **29** (1982) 333–362

[10] Boccalatte, A., Giglio, D., Paolucci, M.: "An Object-Oriented Modeling Approach Based on Entity-Relationship Diagrams and Petri Nets". In: IEEE Internal conference on Systems, Man and Cybernetics, San Diego, CA (1998)

[11] Gogolla, M., Huge, A.K., Randt, B.: "Stepwise Re-Engineering and Development of Object-Oriented Database Schemata". In: International Workshop on Database and Expert Systems Applications, Vienna, Austria (1998)

[12] Bisova, V., Richta, K.: "Transformation of UML Models into XML". In: ADBIS-DASFAA Symposium on Advances in Databases and Information Systems, Prague, Czech Republic (2000)

[13] Conrad, R., Scheffner, D., Freytag, J.C.: "XML Conceptual Modeling using UML". In: International Conference on Conceptual Modeling, Salt Lake City, UT (2000)

[14] Hou, J., Zhang, Y., Kambayashi, Y.: "Object-Oriented Representation for XML Data". In: International Symposium on Cooperative Database Systems for Advanced Applications, Beijing, China (2001)

[15] Al-Jadir, L., El-Moukaddem, F.: "F2/XML: Storing XML Documents in Object Databases". In: International Conference on Object Oriented Infomation Systems, Montpellier, France (2002)

[16] Christophides, V., Abiteboul, S., Cluet, S., Scholl, M.: "From Structured Document to Novel Query Facilities". In: ACM SIGMOD International Conference on Management of Data, Minneapolis, MN (1994)

[17] Batini, C., Ceri, S., Navathe, S.B.: "Conceptual Database Design: An Entity-Relationship Approach". The Benjamin/Cummings Pub. (1992)

[18] Bourret, R.: "XML and Databases". Web page (1999) http://www.rpbourret.com/xml/XMLAndDatabases.htm.

[19] Lee, D., Chu, W.W.: "CPI: Constraints-Preserving Inlining Algorithm for Mapping XML DTD to Relational Schema". Journal of Data & Knowledge Engineering **39** (2001) 3–25

[20] Fernandez, M.F., Tan, W.C., Suciu, D.: "SilkRoute: Trading between Relations and XML". In: International World Wide Web Conference, Amsterdam, Netherlands (2000)

[21] Wood, P.T.: "Optimizing Web Queries Using Document Type Definitions". In: International Workshop on Web Information and Data Management, Kansas City, MO (1999) 28–32

[22] Lee, D., Chu, W.W.: "Comparative Analysis of Six XML Schema Languages". ACM SIGMOD Record **29** (2000) 76–87

[23] Lee, D., Mani, M., Chiu, F., Chu, W.W.: "Nesting-based Relational-to-XML Schema Translation". In: International Workshop on the Web and Databases, Santa Barbara, CA (2001)

[24] Jaeschke, G., Schek, H.J.: "Remarks on the Algebra of Non First Normal Form Relations". In: ACM Symposium on Principles of Database Systems, Los Angeles, CA (1982)

[25] Lee, D., Mani, M., Chiu, F., Chu, W.W.: "NeT & CoT: Translating Relational Schemas to XML Schemas using Semantic Constraints". In: ACM International Conference on Information and Knowledge Management, McLean, VA (2002)

[26] Turau, V.: "Making Legacy Data Accessible for XML Applications". Web page (1999) http://www.informatik.fh-wiesbaden.de/~turau/veroeff.html.

[27] Miller, R.J., Ioannidis, Y.E., Ramakrishnan, R.: "Schema Equivalence in Heterogeneous Systems: Bridging Theory and Practice (Extended Abstract)". In: Extending Database Technology, Cambridge, UK (1994)