Title:
*Translating relational schema with constraints into XML schema*

Authors:
Dr Joseph Fong
Associate Professor
Department of Computer Science
City University of Hong Kong,
Tat Chee Avenue, Hong Kong
email: csifong@cityu.edu.hk

Dr Anthony Fong+,
Associate Professor
Department of Electronic Engineering
City University of Hong Kong,
Tat Chee Avenue, Hong Kong

H.K. Wong*,
Ph.D. Student
Department of Computer Science
City University of Hong Kong,
Tat Chee Avenue, Hong Kong

Philip Yu
Research Assistant
Department of Electronic Engineering
City University of Hong Kong,
Tat Chee Avenue, Hong Kong
Translating relational schema with constraints into XML schema

Abstract
With XML adopted as technology trend on the Internet, and with investment in the current relational database systems, companies must convert their relational data into XML documents for data transmission on the Internet. In the process, to preserve the users’ relational data requirements of data constraints into the converted XML documents, we must define a meaningful root element for each XML document. The construction of an XML document is based on the root element and its relevant elements. The root element can be selected from a relational entity table in the existing relational database, which depends on the requirements to present the business behind. The relevant elements are mapped from the related entities, based on the navigability of the chosen entity. The derived root and relevant elements can form a Data Type Definition Graph (DTD-graph) of an XML conceptual schema diagram which can be mapped into a Data Type Definition (DTD) of an XML schema. The result is a translated XML schema with semantic constraints transferred from a relational conceptual schema of an Extended Entity Relationship (EER) model. The data conversion from relational data to the XML documents can be done after the schema translation. The relational data are loaded into XML documents according to the translated DTD.

Keywords: Relational schema, XML schema, schema translation, extended entity relationship model, document type definition, Document Type Definition Graph, XML view

1 Introduction
XML (eXtensible Markup Language) has emerged and is gradually accepted as the standard for data interchange in the Internet world. XML enabled features packaged by the key Relational Database Vendor in the market as the extender or cartridge to the relational database management system. Interoperation of relational database and XML database involves schema translation. The translated XML schema help sharing business data with other systems, interoperability with incompatible systems, exposing legacy data to applications that use XML, e-commerce, object persistence using XML and content syndication [13]. This paper provides a methodology of translating a relational database schema into XML through EER (extended entity relationship) model. A classification table recovers the data semantics from the relational database into EER model, and then maps them into an XML schema.

However, the only standardised method for creating document type definitions is through the use of markup declarations. What is needed is a method of augmenting the existing set of Data Type Definition (DTD) properties with additional properties to achieve true information understanding. There are ways to accomplish this goal by using XML itself. The XML schema will provide a means of using XML instances to define augmented DTDs. The transformation adopts a reverse engineering approach. It reconstructs the semantic model in the form of an EER model from the logical schema by capturing
user’s knowledge. It then forward engineers the relational database into XML schema as shown in Figure 1.

![Architecture of Re-engineering Relational Database into XML Document](image)

**Figure 1** Architecture of Re-engineering Relational Database into XML Document

It creates an intermediate repository (EER universe) to store the information of the database semantics independently for the following two reasons: Relational database vendors will not compromise with each other. By maintaining a central static environment and providing close interface to both Relational database and XML database, the database reverse engineering approach can have a longer life span.

XML databases use XML documents as fundamental units, define a model such as elements, attributes, PCDATA, etc for an XML instance, and store data as either binary code or text file. XML documents have been widely used on the Internet for business transaction in both B2B and B2C. We expect a strong need to migrate relational databases into XML documents along with their data dependency constraints as follows:

**Functional dependency**: A functional dependency is a statement of the form $X \rightarrow Y$, where $X$ and $Y$ are sets of attributes. The FD: $X \rightarrow Y$ holds for relation $R$ if whenever $s$ and $t$ are tuples of $R$ where $s[X] = t[X]$, then $s[Y] = t[Y]$.

**Inclusion dependency**: An inclusion dependency is a statement of the form $X \subseteq Y$ such that $X$ is a subset of $Y$. For example, $X$ is a foreign key of a child relation and $Y$ is a referred primary key of its parent relation.

**Multi-valued dependency**: Let $R$ be a relation, and let $X$, $Y$, and $Z$ be attributes of $R$. Then $Y$ is multi-dependent on $X$ in MVD: $X \rightarrow \rightarrow Y \mid Z$ if and only if the set of $Y$-values matching a given $(X$-value, $Z$-value) pair in $R$ depends only on the $X$-value and is independent of the $Z$-value.

The rest of the paper is organized as follows: Section 2 gives a discussion of the related work. Section 3 presents the rules to translate a semantically relational schema to XML schema (DTD). Section 4 illustrates a case study and prototype on the hospital database system, to shows the feasibility of proposed methodology and we conclude in section 5. Appendix shows another example with different root selection.
2 Related Works
In recent years, with the growing importance of XML (Extensible Markup Language) documents as a means to represent data in the World Wide Web, there have been a lot of effort on devising new technologies to store and retrieve XML documents using relational database. Methods for storing XML documents in relational databases can roughly be classified into three categories: structure-mapping, model-mapping and semantic-preserving approaches.

2.1 Model-mapping approach
There have been several studies that use fixed relational schemas to store XML document. Such approach is called model-mapping approaches. Each of them has different mapping rules and database schema.

The “Edge” approach describes in [6] stores the XML data as a direct graph / tree in a single relational table. This approach maintains edges individually. Therefore it needs to concatenate the edges to form a path for processing user queries. As a sample table, it only keeps edge-label, rather than the labeled paths. Therefore a large number of joins is needed to check edge connections. Similar to “Edge” approach, in [11], the edge table is enriched by some information in order to distinguish between different target nodes. The content of a document is stored in a leaf value (Leaf table) or in an attribute value (Attr table). Both are referenced from the Edge table via the foreign key. The edges of the document tree are identified by source node and target node. Each document has a unique ID so that an edge can be assigned to one document. A drawback of this approach is that the decomposition of document produces a lot of tuples to be inserted into database. Therefore, the load time may increase for large document. In XRel [9], an XML document is decomposed into nodes on the basis of its tree structure and stored in relational tables according to the node type, with path information from the root to each node. XRel stores the directed graph of an XML document in four tables. The advantage of XRel is that it does not require recursive queries, and can perform the same function within the SQL-92 standard. XParent [3] adopts the data model of XPath to represent XML documents, which models a document as an ordered tree. It uses the similar schema like XRel. The data-path id replaces the start and end pairs used in XRel. The advantage of XParent is that it can be efficiently supported using the conventional index mechanisms like B-tree. One drawback of XParent is that it requires a large number of joins to check edge-connections for processing complex query. In XML-Relational conversion [7], each document is stored in two relational tables. This approach preserves the nested structure of an XML document. A shortcoming of this approach is that PathId depends on an element's tag, it might happen that some elements occur multiple times which violates the definition of primary key (PathId). Extra works need to solve such conflicts.

2.2 Structure-mapping approach
In structure-mapping, schemas such as Document Type Definition (DTD) are extracted from XML documents and a database schema is defined for each XML document. The algorithms proposed
comprising of different methods on how to map attributes and values in XML documents into the relational tables.

Fernández et al. [8] describe a general framework for mapping relational databases to XML virtual views using a declarative query language, RXL (Relational to XML Transformation Language). The operation starts by writing an RXL query that defines the XML virtual view of the database. The main shortcoming is that queries over the views often produce composed queries with many unions. Varlamis and Vazirgiannis [4] develop the X-Database system that acts as an interface between the application and database. The base of the system is an XML-Schema that describes the logical model of interchanged information. A drawback of the X-Database system is that the XML-Schema defined once in the beginning of the process and never be changed, but the schema is change over time in the majority of applications in the real world. XPERANTO [10] operates as a middleware on top on (object-) relational database system. It starts by providing a default virtual view of a given (object-) relational database. User can then create more complex or specialized views based on default view by using an XML query language. One attractive aspect of the XPERANTO approach is that it works in any existing relational database system because XPERANTO system generates regular SQL and tags the results outside the database engine. Carey et al. [2] presents a visual based XML document management system, VXXMLR. First the XML document is parsed into a Document Object Mode tree and the DTD of the document is extracted. Then the document tree is mapped and stored into relational table. VXXMLR maintains some statistics of data and a path directory, which are used in the query rewriting process to reduce the number of SQL statements and simplify join conditions.

2.3 Semantic-preserving approach
Semantic-preserve approach generates an XML structure that is able to describe the semantics and structure in the underlying relational database.

In [12], Du et al. develop a methodology which employs the semantically rich Object-Relational-Attribute model for semi-structured data (ORA-SS) in the translation process. ORA-SS models a rich variety of semantic constraints (strong / weak entities, binary / n-ary / recursive / ISA relationship type, single-valued / multi-valued attributes of entity types or relationship types and cardinality constraints) in the underlying relational database, and represent the implicit structures of relational data using hierarchy and referencing. ORA-SS preserves the inherent semantics and implicit structure in relation schema. In [5], Fong, et al. de-normalize the relational schema into joined tables which are transformed into Document Object Model (DOM) according to their data dependency constrains. These DOMs are integrated into a DOM which is translated into an XML document. The data dependencies constraints in the de-normalized relational schema are mapped into XML document trees in elements and sub-elements. In the process, the partial functional dependencies are mapped into elements and attributes. The transitive data dependencies are mapped into element, sub-element, and sub-sub-element in the XML documents. The multi-valued dependencies are mapped into multiple sub-elements under one element. The join dependencies are mapped into a group element. As a result,
the data semantics in the relational schema are translated and preserved in the XML document. In [1], Duta et al. transform relational schemas into nested-based XML schema for each relational data source. The proposed method preserves the structural constraints (cardinality and participation) of the relationships from the underlying relational database source and representing the flat relation structures in a compact nested XML structure.

3 Methodology of Translating Relational Schema into XML Schema

By following a stepwise procedure in Figure 1, we translate a relational schema into an XML schema based on a selected root element, and then load relational data into an XML document as follows:

**Step 1: Reverse Engineering Relational Schema into an EER Model**

By use of classification tables to define the relationship between keys and attributes in all relations, we can recover their data semantics in the form of an EER model.

**Step 2: Schema Translation from EER Model into DTD-graph and DTD**

We can map the data semantics in the EER model into DTD-graph according to their data dependencies constraints. These constraints can then be transformed into DTD as XML schema as shown in the following:

**Rule 0: Defining a Root Element**

To select a root element, we must put its relevant information into an XML schema. Relevance concerns with the entities that are related to a selected entity by the user. The relevant classes include the selected entity and all its relevant entities that are navigable as follows:

**Navigability:**

In an EER mode, we can navigate from entity to another entity in one-to-many cardinality in correspondence to XML hierarchical containment tree model from parent element to child elements.

**Relevance:**

In an EER model, a superclass entity data occurrences must include its subclass entity data occurrences. On the other hand, a subclass entity may have its own attributes. Thus, a superclass entity requested by the users must include its relevant subclass entity.

Navigability specifies the feasibility of the traversal from an entity to its related entities. Relationship can be directional with navigability. The process is similar to the process when we walk the tree structure of a DTD-graph. We navigate each relationship, then each relationship from the children table of the previous relationship and so on.
In Figure 2, Entity E is the selected entity, and Entities F, G and H are relevant entities because they are navigable from Entity E. The navigable entities in the EER Model are mapped as sub-elements under root elements in a hierarchy structure. All elements are declared as EMPTY in this situation. Each attribute of the relevant entity is mapped into the attribute of the corresponding element. In the example, Entities F, G and H can be mapped into Elements F, G and H in the content model of a translated XML schema.

**Figure 2** Selected Root Element and Relevant Entities and its Mapped DTD-graph

**Rule 1: Mapping Weak Entity from RDB to XML**

A weak entity depends on its strong entity such that the primary key of the weak entity is also a foreign key addressing to the primary key of its strong entity, and cannot be a null value. In DTD, we transform the strong entity into an element with ID and the weak entity into another element which refers to the ID element using IDREF as shown in Figure 3.

**Figure 3** Weak Entity: Schema Translation
The existence dependency constraints can be preserved in the translated XML schema as shown Table 1.

**Table 1 Dependency Constraints Preservation**

<table>
<thead>
<tr>
<th>Data Dependency</th>
<th>Relational Schema</th>
<th>Translated XML DTD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Functional</td>
<td>B.A1 → A.A1</td>
<td>Instance (B) → Instance (A)</td>
</tr>
</tbody>
</table>

**Rule 2: Mapping Participation from RDB to XML**

A child table is in total participation with a parent table provided that all data occurrences of the child table must participate in a relationship with the parent table. A foreign key of a child table in total participation must address to the primary key of its parent table and cannot be a null value. A child table is in partial participation with a parent table provided that the data occurrences of the child table are not totally participated in a relationship with the parent table. A foreign key of a child table in partial participation must address to the primary key of its parent table and can be a null value. In DTD, we translate the total and partial participations into an optional occurrence as shown in Figure 5 and Figure 7.

**Case 1: Total / Mandatory Participation**

**Figure 5 Total Participation: Schema Translation**

**Figure 6 Total Participation: Data Conversion**
Case 2: Partial / Optional Participation

The functional dependency of relational schema is preserved in the translated XML schema where a foreign key determines a referred primary key and an instance of a child element determines a data occurrence of a parent element as shown in Table 2.

<table>
<thead>
<tr>
<th>Data Dependency</th>
<th>Relational Schema</th>
<th>Translated XML DTD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Functional</td>
<td>B.A1 → A.A1</td>
<td>Instance (B) → Instance (A)</td>
</tr>
</tbody>
</table>

Rule 3: Mapping Cardinality from RDB to XML

One-to-one cardinality indicates that a foreign key of a child table addresses to a primary key of a parent table in one to one occurrence. One-to-many cardinality indicates that a primary key of a parent table is addressed by many foreign keys of a child table in one to many occurrence. Many-to-many cardinality indicates that a primary key of a parent table is addressed by many foreign keys of a child table and vice versa. This pair of tables are thus in many to many cardinality. In DTD, we translate one-to-one cardinality into parent and child element (Figure 9) and one-to-many cardinality into parent and child element with multiple occurrences (Figure 11). In many-to-many cardinality, it is mapped into DTD of a hierarchy structure with ID and IDREF as shown in Figure 13.
Case 1: One-to-one Cardinality

![Image of EER Model and DTD Graph]

**Figure 9** One-to-one Cardinality: Schema Translation

**Case 2: One-to-many Cardinality**

![Image of EER Model and DTD Graph]

**Figure 11** One-to-many Cardinality: Schema Translation

**Figure 10** One-to-one Cardinality: Data Conversion

**Figure 12** One-to-many Cardinality: Data Conversion
Case 3: Many-to-many Cardinality

The functional dependency and multi-valued dependency of relational schema are preserved in the translated XML schema as shown in Table 3.

<table>
<thead>
<tr>
<th>Case</th>
<th>Data dependency</th>
<th>Relational Schema</th>
<th>Translated XML DTD</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 &amp; 2</td>
<td>Functional</td>
<td>B.A1 → A.A1</td>
<td>Instance (B) → Instance (A)</td>
</tr>
<tr>
<td>3</td>
<td>Multi-valued</td>
<td>A.A1 → R.A1, B.B1→ R.B1</td>
<td>Instance (A) → Instance (R), Instance (B) → Instance (R)</td>
</tr>
</tbody>
</table>

Rule 4: Mapping Aggregation from RDB to XML

An aggregation specifies a whole-part relationship between an aggregate such that a class represents the whole and a constituent represents part. DTD can construct part-of relationship in the element content. For example, in Figure 15, Entity B, Entity C and relationship R1 form an aggregate entity which is related to another Entity A. They can be mapped into DTD as follows:
The functional dependency of relational schema is preserved in the translated XML schema as shown in Table 4.

### Table 4 Dependency Constraints Preservation

<table>
<thead>
<tr>
<th>Data Dependency</th>
<th>Relational Schema</th>
<th>Translated XML DTD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Functional</td>
<td>R1.A1→A.A1</td>
<td>Instance (R1) → Instance (A)</td>
</tr>
</tbody>
</table>

#### Rule 5: Mapping ISA Relationship from RDB to XML

The isa defines as relationship between a subclass entity to a superclass entity. In DTD, we transform each subclass entity as a child element which refers to its parent element such that each parent element can have zero to one child elements as:
The functional dependency constraints of relational schema can be preserved in the translated XML schema as shown in Table 5.

**Table 5 Dependency Constraints Preservation**

<table>
<thead>
<tr>
<th>Data Dependency</th>
<th>Relational Schema</th>
<th>Translated XML DTD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Functional</td>
<td>B.A1 → A.A1</td>
<td>Instance(B) → Instance (A)</td>
</tr>
</tbody>
</table>

**Rule 6: Mapping Generalization from RDB to XML**

The generalization defines a relationship between entities to build a taxonomy of classes: One entity is a more general description of a set of other entities. In DTD, we transform the general superclass entity into an element, the element type originating from the superclass. For example, in Figure 19 and Figure 21, we present the generalization of Entity B and Entity C into Entity A in DTD.

**Case 1: Disjoint Generalization**
Figure 19 Disjoint Generalization: Schema Translation

Figure 20 Disjoint Generalization: Data Conversion

Case 2: Overlap Generalization

Figure 21 Overlap Generalization: Schema Translation

Figure 22 Overlap Generalization: Data Conversion
The functional dependency constraints of relational schema can be preserved in the translated XML schema as shown in Table 6.

### Table 6 Dependency Constraints Preservation

<table>
<thead>
<tr>
<th>Data Dependency</th>
<th>Relational Schema</th>
<th>Translated XML DTD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Functional</td>
<td>B.A1 → A.A1</td>
<td>Instance(B) → Instance (A)</td>
</tr>
<tr>
<td></td>
<td>C.A1 → A.A1</td>
<td>Instance (C) → Instance (A)</td>
</tr>
</tbody>
</table>

### Rule 7: Mapping Categorization from RDB to XML

A subclass table is a subset of a categorization of its superclass tables. In other words, a subclass table is a subset of a union superclass tables such that the data occurrence of a subclass table must appear in one and only one superclass table. In DTD, we transform the supper classes into elements, and their common subclass into an element on the same level. Each element receives an additional “artificial” ID attribute declared as #REQUIRED referred by their common element’s IDREF in DTD as shown in Figure 23.

**Figure 23** Categorization: Schema Translation

**Figure 24** Categorization: Data Conversion
The inclusion dependency constraints of relational schema can be preserved in the translated XML schemas shown in Table 7.

### Table 7 Dependency Constraints Preservation

<table>
<thead>
<tr>
<th>Data Dependency</th>
<th>Relational Schema</th>
<th>Translated XML DTD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inclusion</td>
<td>A.A1 ⊆ (B.A1 ∪ C.A1)</td>
<td>Instance(A) ⊆ (Instance(B) ∪ Instance(C))</td>
</tr>
</tbody>
</table>

**Rule 8: Mapping N-ary Relationship from RDB to XML**

Multiple tables relate to each other in an n-ary relationship. An n-ary relationship is a relationship relation for multiple tables such that components of the former's compound primary key addressing to the primary keys of the latter which are related to each other. In DTD, we transform n-ary relationship into group of element as shown in Figure 25.

---

**Figure 25 N-ary Relationship: Schema Translation**

**Figure 26 N-ary Relationship: Data Conversion**
The functional dependency constraints of relational schema can be preserved in the translated XML schema as shown in Table 8.

<table>
<thead>
<tr>
<th>Data Dependency</th>
<th>Relational Schema</th>
<th>Translated XML DTD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Functional</td>
<td>R.A1 → A.A1</td>
<td>Instance Group → Instance (A)</td>
</tr>
<tr>
<td></td>
<td>R.B1 → B.B1</td>
<td>+ Instance(B) + Instance (C)</td>
</tr>
<tr>
<td></td>
<td>R.C1 → C.C1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>R.A1, R.B1, R.C1 → 0</td>
<td></td>
</tr>
</tbody>
</table>

**Table 8 Dependency Constraints Preservation**

**Step 3 Data Conversion from Relational Database to XML Documents**

As the result of the schema translation in step 2, we translate an EER model into different views of XML schemas based on their selected root elements. For each translated XML schema, we can read its corresponding source relation sequentially by embedded SQL, that is, one tuple at one time, starting a parent relation. The tuple can then be loaded into XML document according to the mapped XML DTD. Then we read the corresponding child relation tuple(s), and load them into XML document. The procedure is to process corresponding parent and child relations in the source relational database according to the translated parent and child elements in the mapped DTD and so on as shown in Figure 27.

```begin
begin
    while not end of element do
        read an element from the translated target DTD;
        read the tuple of a corresponding relation of the element from the source relational database;
        read the child elements of the element according to the DTD;
        while not at end of the corresponding child relation in the source relational database do
            read the tuple from the child relation such that the child's corresponding to the processed parent relation's tuple;
            load the tuple to the target XML document;
            end loop // end inner loop
        end loop // end outer loop
    end loop
end
```

**Figure 27 Data Conversion Algorithm**

**4 Case Study**

Consider a case study of a Hospital Database System. In this system, a patient can have many record folders. Each record folder can contain many different medical records of the patient. The AE, a ward and an outpatient record can be generalized as a medical record. A country has many patients. A borrower of the record folder of the patient can be a department, a doctor or other hospital for their references or checking. Once a record folder is borrowed, a loan history is created to record the details about it. The relational schemas for this case study are shown below. Notice that underlined and italic means primary key and * means foreign key.

Relation Country
<table>
<thead>
<tr>
<th>Country_No</th>
<th>Country_Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>C0001</td>
<td>China</td>
</tr>
<tr>
<td>C0002</td>
<td>Canada</td>
</tr>
<tr>
<td>C0003</td>
<td>Korea</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Relation Patient</strong></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>HKID</td>
<td>Patient_Name</td>
</tr>
<tr>
<td>------</td>
<td>--------------</td>
</tr>
<tr>
<td>E3766849</td>
<td>Smith</td>
</tr>
<tr>
<td>E8018229</td>
<td>Bloor</td>
</tr>
<tr>
<td>E6077888</td>
<td>Kim</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Relation Record_Folder</strong></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Folder_No</td>
<td>Location</td>
</tr>
<tr>
<td>-----------</td>
<td>----------</td>
</tr>
<tr>
<td>F_21</td>
<td>Hong Kong</td>
</tr>
<tr>
<td>F_22</td>
<td>Kowloon</td>
</tr>
<tr>
<td>F_23</td>
<td>New Territories</td>
</tr>
<tr>
<td>F_24</td>
<td>New Territories</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Relation AE_Record</strong></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>*Medical_Rec_No</td>
<td>AE_No</td>
</tr>
<tr>
<td>-----------------</td>
<td>------</td>
</tr>
<tr>
<td>M_352001</td>
<td>AE_1</td>
</tr>
<tr>
<td>M_362001</td>
<td>AE_2</td>
</tr>
<tr>
<td>M_333333</td>
<td>AE_3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Relation Medical_Record</strong></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Medical_Rec_No</td>
<td>Create_Date</td>
</tr>
<tr>
<td>----------------</td>
<td>-------------</td>
</tr>
<tr>
<td>M_311999</td>
<td>Jan-1-1999</td>
</tr>
<tr>
<td>M_322000</td>
<td>Nov-12-1998</td>
</tr>
<tr>
<td>M_331998</td>
<td>Nov-10-1998</td>
</tr>
<tr>
<td>M_341999</td>
<td>Dec-20-1999</td>
</tr>
<tr>
<td>M_352001</td>
<td>Jan-15-2001</td>
</tr>
<tr>
<td>M_362001</td>
<td>Feb-01-2001</td>
</tr>
<tr>
<td>M_382001</td>
<td>Feb-22-2001</td>
</tr>
<tr>
<td>M_333333</td>
<td>Mar-03-01</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Relation Ward_Record</strong></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>*Medical_Rec_No</td>
<td>Ward_No</td>
</tr>
<tr>
<td>-----------------</td>
<td>---------</td>
</tr>
<tr>
<td>M_311999</td>
<td>W_41</td>
</tr>
<tr>
<td>M_322000</td>
<td>W_43</td>
</tr>
</tbody>
</table>
**Relation Outpatient_Record**

<table>
<thead>
<tr>
<th>Medical_Rec_No</th>
<th>Outpatient_No</th>
<th>Specialty</th>
</tr>
</thead>
<tbody>
<tr>
<td>M_331998</td>
<td>O_51</td>
<td>Heart</td>
</tr>
<tr>
<td>M_341999</td>
<td>O_52</td>
<td>Ophthalmic</td>
</tr>
<tr>
<td>M_382001</td>
<td>O_53</td>
<td>Therapy</td>
</tr>
</tbody>
</table>

**Relation Borrower**

<table>
<thead>
<tr>
<th>Borrower_No</th>
<th>Borrower_Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>B1</td>
<td>Bloor</td>
</tr>
<tr>
<td>B2</td>
<td>Smith</td>
</tr>
<tr>
<td>B3</td>
<td>Kim</td>
</tr>
<tr>
<td>B11</td>
<td>X-Ray</td>
</tr>
<tr>
<td>B12</td>
<td>Infant</td>
</tr>
<tr>
<td>B14</td>
<td>Skin</td>
</tr>
<tr>
<td>B21</td>
<td>Mac Neal</td>
</tr>
<tr>
<td>B22</td>
<td>Riveredge</td>
</tr>
<tr>
<td>B25</td>
<td>Golden Park</td>
</tr>
</tbody>
</table>

**Relation Borrow**

<table>
<thead>
<tr>
<th>Borrower_No</th>
<th>Folder_No</th>
</tr>
</thead>
<tbody>
<tr>
<td>B1</td>
<td>F_21</td>
</tr>
<tr>
<td>B1</td>
<td>F_22</td>
</tr>
<tr>
<td>B2</td>
<td>F_22</td>
</tr>
<tr>
<td>B3</td>
<td>F_23</td>
</tr>
<tr>
<td>B11</td>
<td>F_21</td>
</tr>
<tr>
<td>B12</td>
<td>F_22</td>
</tr>
<tr>
<td>B14</td>
<td>F_23</td>
</tr>
<tr>
<td>B21</td>
<td>F_21</td>
</tr>
<tr>
<td>B22</td>
<td>F_21</td>
</tr>
<tr>
<td>B22</td>
<td>F_24</td>
</tr>
<tr>
<td>B25</td>
<td>F_23</td>
</tr>
</tbody>
</table>

**Relation Loan_History**

<table>
<thead>
<tr>
<th>Borrower_No</th>
<th>Folder_No</th>
<th>Loan_Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>B1</td>
<td>F_21</td>
<td>Jan-10-2002</td>
</tr>
<tr>
<td>B1</td>
<td>F_22</td>
<td>Jan-10-2002</td>
</tr>
<tr>
<td>B2</td>
<td>F_22</td>
<td>Sep-29-2002</td>
</tr>
<tr>
<td>B3</td>
<td>F_23</td>
<td>Sep-29-2002</td>
</tr>
<tr>
<td>Borrower No</td>
<td>Department Name</td>
<td>Date</td>
</tr>
<tr>
<td>-------------</td>
<td>-----------------</td>
<td>------------</td>
</tr>
<tr>
<td>B11</td>
<td>X-Ray</td>
<td>Jun-12-2002</td>
</tr>
<tr>
<td>B12</td>
<td>Infant</td>
<td>Jan-07-2002</td>
</tr>
<tr>
<td>B13</td>
<td>Chest</td>
<td>Jan-11-2002</td>
</tr>
<tr>
<td>B14</td>
<td>Skin</td>
<td>Feb-01-2002</td>
</tr>
<tr>
<td>B15</td>
<td>Therapy</td>
<td>Mar-03-2002</td>
</tr>
</tbody>
</table>

### Relation Department

<table>
<thead>
<tr>
<th>Borrower No</th>
<th>Department Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>B11</td>
<td>X-Ray</td>
</tr>
<tr>
<td>B12</td>
<td>Infant</td>
</tr>
<tr>
<td>B13</td>
<td>Chest</td>
</tr>
<tr>
<td>B14</td>
<td>Skin</td>
</tr>
<tr>
<td>B15</td>
<td>Therapy</td>
</tr>
</tbody>
</table>

### Relation Doctor

<table>
<thead>
<tr>
<th>Borrower No</th>
<th>Doctor Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>B1</td>
<td>Bloor</td>
</tr>
<tr>
<td>B2</td>
<td>Smith</td>
</tr>
<tr>
<td>B3</td>
<td>Kim</td>
</tr>
<tr>
<td>B4</td>
<td>Chitson</td>
</tr>
<tr>
<td>B5</td>
<td>Navathe</td>
</tr>
</tbody>
</table>

### Relation Other Hospital

<table>
<thead>
<tr>
<th>Borrow No</th>
<th>Hospital Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>B21</td>
<td>Mac Neal</td>
</tr>
<tr>
<td>B22</td>
<td>Riveredge</td>
</tr>
<tr>
<td>B23</td>
<td>Stone Town</td>
</tr>
<tr>
<td>B24</td>
<td>North Community</td>
</tr>
<tr>
<td>B25</td>
<td>Golden Park</td>
</tr>
</tbody>
</table>

By following the stepwise procedures that were mentioned before, we now translate this relational schema into the XML Schema of DTD as shown below:

**Step 1: Reverse Engineering Relational Schema into an EER Model**

By using classification table, we can recover the EER model from the given relational schemas as shown in Figure 28.
Step 2.1: Defining Root Element

In this case study, suppose we concern the patient medical records, so the entity Patient is selected. Then we define a meaningful name for the root element, called Patient_Records. All patients are under the root element as shown in below.

*XML Schemas (DTD):*

```xml
<!ELEMENT Patient_Records (Patient)+>
```

We start from the entity Patient in the EER model and then find the relevant entities for it. The relevant entities include the related entities that are navigable from the parent entity. Entities Record Folder, Medical Record and Borrow are considered relevant entities because they are navigable from the entity Patient. Since the relationship between the entity Patient and the entity Country is many-to-one, then the entity County is considered not navigable from the entity Patient according to our methodology. As a result, a DTD-graph that starting from the entity Patient is formed and shown in Figure 29.
Entity Patient is a direct child of the root element, Patient_Records. Since the entities Record Folder and Medical Record are navigable from the Patient entity, then we map all those entities into the elements of the XML DTD. We then define the attributes of those elements by using the definition of the relational schema as shown below:

**Listing 1 Relational Schema**

```sql
Patient (HKID, Patient_Name, *Country_Code)
Record_Folder (Folder_No, Location, *HKID)
Medical_Record (Medical_Rec_No, Create_Date, Sub_Type, *Folder_No)
```

**Listing 2 Translated XML Schema in DTD**

```xml
<!ELEMENT Patient_Records (Patient) +>
<!ELEMENT Patient (Record_Folder)>
<!ELEMENT Record_Folder (Medical_Record)>
<!ELEMENT Medical_Record EMPTY>
<!ATTLIST Patient HKID CDATA #REQUIRED>
<!ATTLIST Patient Patient_Name CDATA #REQUIRED>
<!ATTLIST Patient Country_Code CDATA #REQUIRED>
<!ATTLIST Record_Folder Folder_No CDATA #REQUIRED>
<!ATTLIST Record_Folder Location CDATA #REQUIRED>
<!ATTLIST Record_Folder HKID CDATA #REQUIRED>
<!ATTLIST Medical_Record Medical_Rec_No CDATA #REQUIRED>
<!ATTLIST Medical_Record Create_Date CDATA #REQUIRED>
<!ATTLIST Medical_Record Sub_Type CDATA #REQUIRED>
```
Step 2.2: Mapping Weak Entity into Content Model
It is not applicable in this step.

Step 2.3: Mapping Participation into Content Model
The relationship between the entities Patient and the Record Folder is total participation. The relationship between the entities Record Folder and the Medical Record is also in total participation. Therefore, the content model of the XML schema is translated as shown below. Notice that all foreign keys in relational schema will not be mapped into XML DTD because they will be represented in containment or ID and IDREF.

Listing 3 Relational Schema
Patient (HKID, Patient_Name, *Country_Code)
Record_Folder (Folder_No, Location, *HKID)
Medical_Record (Medical_Rec_No, Create_Date, Sub_Type, *Folder_No)

Listing 4 Translated XML Schema
<!ELEMENT Patient (Record_Folder*)>
<!ELEMENT Record_Folder (Medical_Record*)>
<!ELEMENT Medical_Record EMPTY>
<!ATTLIST Patient HKID CDATA #REQUIRED>
<!ATTLIST Patient Patient_Name CDATA #REQUIRED>
<!ATTLIST Patient Country_Code CDATA #REQUIRED>
<!ATTLIST Record_Folder Folder_No CDATA #REQUIRED>
<!ATTLIST Record_Folder Location CDATA #REQUIRED>
<!ATTLIST Medical_Record Medical_Rec_No CDATA #REQUIRED>
<!ATTLIST Medical_Record Create_Date CDATA #REQUIRED>
<!ATTLIST Medical_Record Sub_Type CDATA #REQUIRED>

Step 2.4: Mapping Cardinality into Content Model
The relationship between entities Borrower and entity Record_Folder is in many-to-many cardinality. It is because a borrower can borrow many record folders and a record folder can be borrowed by many borrowers. In this many-to-many cardinality, we will not include the relationship between entities borrow and borrower since they are in many-to-one relationship. The translated XML schema together with the many-to-many relationship is shown below:

Listing 5 Relational Schema
Record_Folder (Folder_No, Location, *HKID)
Listing 6 Translated XML Schema

<!ELEMENT Record_Folder (Borrow*, Medical_Record*)>
<!ELEMENT Medical_Record EMPTY>
<!ELEMENT Borrow EMPTY>
<!ATTLIST Borrow Borrower_No CDATA #REQUIRED>

Since the entity Loan_History is also navigable from the Borrow entity and they are in one-to-many relationship, so the modified XML schema will be:

Listing 7 Relational Schema

Loan_History (*Borrow_No, *Folder_No, Loan_Date)

Listing 8 Translated XML Schema

<!ELEMENT Borrow (Loan_History*)>
<!ELEMENT Loan_History EMPTY>
<!ATTLIST Loan_History Folder_No CDATA #REQUIRED>
<!ATTLIST Loan_History Loan_Date CDATA #REQUIRED>

Step 2.5: Mapping Aggregation into Content Model
It is not applicable in this case study.

Step 2.6 Mapping ISA into Content Model
It is not applicable in this case study.

Step 2.7: Mapping Generalization into Content Model
Since the medical record can be an AE, a ward or an outpatient record, so it is a disjoint generalization. Then the translated DTD for the entity Medical Record is shown below:

Listing 9 Relational Schema

Medical_Record (Medical_Rec_No, Create_Date, Sub_Type, *Folder_No)
AE_Record (*Medical_Rec_No, AE_No)
Ward_Record (*Medical_Rec_No, Ward_No, Admission_Date, Discharge_Date)
Outpatient_Record (*Medical_Rec_No, Outpatient_No, Specialty)

Listing 10 Translated XML Schema

<!ELEMENT Medical_Record (AE | Ward | Outpatient)>
<!ATTLIST Medical_Record Medical_Rec_No CDATA #REQUIRED>
Step 2.8: Mapping Categorization into Content Model
Although there is a categorization in this case study, it is not navigable from the entity Patient. Thus it is not applicable.

Step 2.9: Mapping N-ary Relationship into Content Model
It is not applicable in this case study.

As a result, the final XML DTD is shown in Listing 7.11.

**Listing 11** Patient Records DTD

```xml
<!ELEMENT Patient_Records (Patient+)>

<!ELEMENT Patient (Record_Folder*)>
<!ATTLIST Patient
HKID   CDATA  #REQUIRED
Patient_Name  CDATA  #REQUIRED>
Country_No  CDATA  #REQUIRED>

<!ELEMENT Record_Folder (Borrow*, Medical_Record*)>
<!ATTLIST Record_Folder
Folder_No   CDATA  #REQUIRED
Location    CDATA  #REQUIRED>

<!ELEMENT Borrow (Loan_History*)>
<!ATTLIST Borrow
Borrower_No   CDATA  #REQUIRED>

<!ELEMENT Loan_History EMPTY>
```
Step 3 Data Conversion from Relational Database to XML Document

As a result of schema translation in step 2, we can load relational data into an XML document as follows:

**Listing 12** Example of Patient Medical Records XML Document

```xml
<Patient_Records>
    <Patient Country_No="C0001" HKID="E3766849" Patient_Name="Smith">
        <Record_Folder Folder_No="F_21" Location="Hong Kong">
            <Borrow Borrower_No="B1">
                <Loan_History Loan_Date="Jan-10-2002" />
            </Borrow>
            <Borrow Borrower_No="B11">
                <Loan_History Loan_Date="Jun-12-2002" />
            </Borrow>
            <Borrow Borrower_No="B21">
                <Loan_History Loan_Date="Feb-01-2002" />
            </Borrow>
        </Record_Folder>
    </Patient>
</Patient_Records>
```
<Patient Country_No="C0001" HKID="E8018229" Patient_Name="Bloor">

</Patient>
Although we have defined the finalized DTD, we need to show the same content between relational database and XML database as a result of schema translation without loss of information when querying the data. We input the same data set to an XML-based information server namely, Tamino by Software AG. Before we input the sample data to the Tamino’s XML database, we have to create a
Tamino’s Schema by using the finalized DTD that we have created for this case study. After we have created all the relational schemas in SQL Server 2000 and the Tamino’ XML database by using the sample data, we can perform some data queries on them:

- To query who has borrowed the Folder Number of ‘F_21’,

We compare the source relational data by executing the following queries in the MS-SQL Server:

```
SELECT * 
FROM Loan_History 
WHERE Folder_No = 'F_21';
```

In the translated XML document, we can type:

```
Patient_Records/Patient/Borrow [@Folder_No = 'F_21']
```

The Tamino’s query result with the same content as relational data is shown below:

Figure 30 Query Result (SQL Server)

Figure 31 Query Result (Tamino)
5 Conclusions

A methodology has been designed to map a relational database schema to an XML schema for Relational database and XML database interoperability. The schema translation procedure is provided with steps and mapping rules to recover the data constraints semantics of relational database into an Extended Entity Relationship model and then mapped into XML schema. The target XML schema is presented in Document Type Definition. Semantic constraints of the relational schema in the form of functional dependencies, inclusion dependencies and multi-valued dependencies are preserved in the translated XML schema. The translation is constructed through an extracted XML view of relational schema, which is based on an selection of its root element and its relevant elements, an entity and its navigable entities chosen by the user, to fulfill his/her data requirement of an XML document.

This paper proposed a conversion method from relational model to XML with data semantics preservation. The differences from other approaches are:

- Generate an XML structure that is able to describe the semantics and structure in the underlying relational database.
- Allow the translation of a set of related relations instead of simple single relation / relationship conversion.
- Obtain properly structure XML data without unnecessary redundancies and proliferation of disconnect XML elements.
- Provide a solution that permits to generate a semantic rich DTD from a relational schema.
- Preserve structural constraints (functional, inclusion and multi-valued dependencies) of the relationships from the input database.
- Represent the flat relational structures in a compact nested XML structure.
- Model multiple relationships that involve a set of relations.

The contribution of this paper includes feasibility of converting data between relational and XML with data semantic preservation. The application of the methodology is B2B and B2C data exchange for the information highway on the Internet.

Table 9 Comparison between Our Approach and Others

<table>
<thead>
<tr>
<th></th>
<th>Our Approach</th>
<th>Other Approach</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source relationship</td>
<td>Include all semantics</td>
<td>Include partial/no semantics</td>
</tr>
<tr>
<td>Semantics aspect</td>
<td>Preserved</td>
<td>Partially/no preserved</td>
</tr>
<tr>
<td>Structured aspect</td>
<td>Preserved</td>
<td>Partially preserved</td>
</tr>
<tr>
<td>Use of Schema</td>
<td>Compulsory</td>
<td>Optional</td>
</tr>
<tr>
<td>XML construction</td>
<td>Use of relational schema</td>
<td>Don’t use of relational schema</td>
</tr>
<tr>
<td>Error &amp; incorrect assumption</td>
<td>Exclusive</td>
<td>Inclusive</td>
</tr>
<tr>
<td>Relational table</td>
<td>Normalized</td>
<td>Unnormalized</td>
</tr>
<tr>
<td>Relations involved (XML</td>
<td>Set of related relations</td>
<td>Single/few relations</td>
</tr>
<tr>
<td>construction)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>XML data redundancy</td>
<td>No</td>
<td>Yes</td>
</tr>
</tbody>
</table>
6. Reference


Appendix – Example: Select another root element

A.1 Defining Root Element

Suppose we concern the borrow information, so the entity Borrow is selected. Then we define a meaningful name for the root element, called Borrow_info. All borrow information are under the root element as shown in below.

XML Schemas (DTD):

<!ELEMENT Borrow_info (Record_Folder, Borrow, Borrower)+>

We start from the entity Borrow in the EER model and then find the relevant entities for it. The relevant entities include the related entities that are navigable from the parent entity. Entities Record Folder, Medical Record and Borrower, Loan_History are considered relevant entities because they are navigable from the entity Borrow. As a result, a DTD-graph that starting from the entity Borrow is formed and shown in Figure A.1.

Entity Borrow is a direct child of the root element, Borrow_info. Since the entities Record Folder, Medical Record, Loan History and Borrower are navigable from the Borrow entity, then we map all those entities into the elements of the XML DTD. We then define the attributes of those elements by using the definition of the relational schema as shown below:

Listing A.1 Relational Schema

Borrow(*Borrower_No, *Folder_No)
Record_Folder (Folder_No, Location, *HKID)
Medical_Record (Medical_Rec_No, Create_Date, Sub_Type, *Folder_No)
Loan_History (*Borrower_No, *Folder_No, Loan_Date)
Borrower(*Borrower_No, Borrower_Name)

Listing A.2 Translated XML Schema in DTD

<!ELEMENT Borrow_info(Record_Folder, Borrow, Borrower)>
<!ELEMENT Record_Folder (Medical_Record)>
<!ELEMENT Medical_Record EMPTY>
A.2 Mapping Weak Entity into Content Model

Entity Loan History is a direct child of the Borrow and it is weak entity, then we map all those entities into the elements of the XML DTD. We then define the attributes of those elements by using the definition of the relational schema as shown below:

**Listing A.3** Relational Schema

Borrow(*Borrower_No, *Folder_No)
Loan_History (*Borrower_No, *Folder_No, Loan_Date)

**Listing A.4** Translated XML Schema in DTD

<!ELEMENT Borrow (Loan_History)>  
<!ELEMENT Loan_History EMPTY>  
<!ATTLIST Borrow Borrower_No CDATA #REQUIRED>  
<!ATTLIST Borrow Folder_No CDATA #REQUIRED>  
<!ATTLIST Loan_History Loan_Date CDATA #REQUIRED>  
<!ATTLIST Loan_History Loan_History_idref idref #REQUIRED>  

A.3 Mapping Participation into Content Model

The relationship between the entities Record Folder and the Medical Record is also in total participation. Therefore, the content model of the XML schema is translated as shown below.
Listing A.5 Relational Schema
Record_Folder (Folder_No, Location, *HKID)
Medical_Record (Medical_Rec_No, Create_Date, Sub_Type, *Folder_No)

Listing A.6 Translated XML Schema

```xml
<!ELEMENT Record_Folder (Medical_Record*)>
<!ELEMENT Medical_Record EMPTY>
<!ATTLIST Record_Folder Folder_No CDATA #REQUIRED>
<!ATTLIST Record_Folder Location CDATA #REQUIRED>
<!ATTLIST Medical_Record Medical_Rec_No CDATA #REQUIRED>
<!ATTLIST Medical_Record Create_Date CDATA #REQUIRED>
<!ATTLIST Medical_Record Sub_Type CDATA #REQUIRED>
```

A.4 Mapping Cardinality into Content Model
The relationship between entities Borrower and entity Record_Folder is in many-to-many cardinality. It is because a borrower can borrow many record folders and a record folder can be borrowed by many borrowers. The translated XML schema together with the many-to-many relationship is shown below:

Listing A.7 Relational Schema
Record_Folder (Folder_No, Location, *HKID)
Borrow (*Borrower_No, *Folder_No)
Borrower(*Borrower_No, Borrower_Name)

Listing A.8 Translated XML Schema

```xml
<!ELEMENT Borrow_info(Record_Folder, Borrow, Borrower)>
<!ELEMENT Record_Folder EMPTY>
<!ELEMENT Borrow EMPTY>
<!ELEMENT Borrower EMPTY>
<!ATTLIST Record_Folder Folder_No CDATA #REQUIRED>
<!ATTLIST Record_Folder Location CDATA #REQUIRED>
<!ATTLIST Record_Folder HKID CDATA #REQUIRED>
<!ATTLIST Record_Folder Record_Folder_id ID #REQUIRED>
<!ATTLIST Borrow Borrower_No CDATA #REQUIRED>
<!ATTLIST Borrow Folder_No CDATA #REQUIRED>
<!ATTLIST Borrow Record_Folder_idref IDREF #REQUIRED>
<!ATTLIST Borrower Borrower_idref IDREF #REQUIRED>
<!ATTLIST Borrower No Borrower_No CDATA #REQUIRED>
<!ATTLIST Borrower Borrower_Name CDATA #REQUIRED>
<!ATTLIST Borrower Borrower_id ID #REQUIRED>
```
Since the entity Loan_History is also navigable from the Borrow entity and they are in one-to-many relationship, so the modified XML schema will be:

**Listing A.9 Relational Schema**

Loan_History (*Borrower_No, *Folder_No, Loan_Date)

**Listing A.10 Translated XML Schema**

```xml
<!ELEMENT Borrow (Loan_History)>  
<!ELEMENT Loan_History EMPTY>  
<!ATTLIST Borrow Borrower_No CDATA #REQUIRED>  
<!ATTLIST Borrow Folder_No CDATA #REQUIRED>  
<!ATTLIST Borrow Borrow_id id #REQUIRED>  
<!ATTLIST Loan_History Folder_No CDATA #REQUIRED>  
<!ATTLIST Loan_History Loan_Date CDATA #REQUIRED>  
<!ATTLIST Loan_History Loan_History_idref idref #REQUIRED>
```

Since the entity Medical Record is also navigable from the Record Folder entity and they are in one-to-many relationship, so the modified XML schema will be:

**Listing A.11 Relational Schema**

Record_Folder (Folder_No, Location, *HKID)  
Medical_Record (Medical_Rec_No, Create_Date, Sub_Type, *Folder_No)

**Listing A.12 Translated XML Schema**

```xml
<!ELEMENT Record_Folder (Medical_Record)>  
<!ELEMENT Medical_Record EMPTY>  
<!ATTLIST Record_Folder Folder_No CDATA #REQUIRED>  
<!ATTLIST Record_Folder Location CDATA #REQUIRED>  
<!ATTLIST Record_Folder HKID CDATA #REQUIRED>  
<!ATTLIST Record_Folder Record_Folder_id ID #REQUIRED>  
<!ATTLIST Medical_Record Medical_Rec_No CDATA #REQUIRED>  
<!ATTLIST Medical_Record Create_Date CDATA #REQUIRED>  
<!ATTLIST Medical_Record Sub_Type CDATA #REQUIRED>
```

**A.5 Mapping Aggregation into Content Model**

It is not applicable in this case study.

**A.6 Mapping ISA into Content Model**
It is not applicable in this case study.

**A.7 Mapping Generalization into Content Model**

Since the medical record can be an AE, a ward or an outpatient record, so it is a disjoint generalization. Then the translated DTD for the entity Medical Record is shown below:

**Listing A.13 Relational Schema**

Medical_Record (Medical_Rec_No, Create_Date, Sub_Type, *Folder_No)
AE_Record (*Medical_Rec_No, AE_No)
Ward_Record (*Medical_Rec_No, Ward_No, Admission_Date, Discharge_Date)
Outpatient_Record (*Medical_Rec_No, Outpatient_No, Specialty)

**Listing A.14 Translated XML Schema**

```xml
<!ELEMENT Medical_Record (AE | Ward | Outpatient)>  
<!ATTLIST Medical_Record Medical_Rec_No CDATA #REQUIRED>  
<!ATTLIST Medical_Record Create_Date CDATA #REQUIRED>  
<!ATTLIST Medical_Record Sub_Type CDATA #REQUIRED>  
<!ELEMENT AE EMPTY>  
<!ATTLIST AE AE_No CDATA #REQUIRED>  
<!ELEMENT Ward EMPTY>  
<!ATTLIST Ward Ward_No CDATA #REQUIRED>  
<!ATTLIST Ward Admission_Date CDATA #REQUIRED>  
<!ATTLIST Ward Discharge_Date CDATA #REQUIRED>  
<!ELEMENT Outpatient EMPTY>  
<!ATTLIST Outpatient Outpatient_No CDATA #REQUIRED>  
<!ATTLIST Outpatient Specialty CDATA #REQUIRED>
```

**A.8 Mapping Categorization into Content Model**

Although there is a categorization in this case study, it is not navigable from the entity Patient. Thus it is not applicable.

**A.9 Mapping N-ary Relationship into Content Model**

It is not applicable in this case study.

As a result, the final XML DTD is shown in Listing A.15.

**Listing A.15 Borrow information DTD**

```xml
<!ELEMENT Borrow_info(Record_Folder, Borrow, Borrower)+>  
<!ELEMENT Record_Folder (Medical_Record)>  
<!ELEMENT Medical_Record (AE | Ward | Outpatient)>
Listing A.16 Example of Borrow information XML Document

```xml
<Borrow_info>
  <Record_Folder Folder_No="F_21" Location="Hong Kong" HKID="E3766849" Record_Folder_idref="RF01">
    <Medical_Record Medical_Rec_No="M_311999" Create_Date="Jan-1-1999" Sub_Type="W">
      <Ward_Record Ward_No="W_41" Admission_Date="Jan-1-1999" Discharge_Date="Mar-20-1999"/>
    </Medical_Record>
  </Record_Folder>
  <Medical_Record Medical_Rec_No="M_322000" Create_Date="Nov-12-1998" Sub_Type="W">
    <Ward_Record Ward_No="W_43" Admission_Date="Nov-12-1998" Discharge_Date="Dec-14-1998"/>
  </Medical_Record>
</Borrow_info>
```
<Medical_Record Medical_Rec_No="M_362001" Create_Date="Feb-1-2001" Sub_Type="A">
  <AE_Record AE_No="AE_2"/>
</Medical_Record>
</Record_Folder>

<Borrow Borrower_No="B22" Folder_No="F_21" Borrow_id="BW09" Record_Folder_idref="RF01" Borrower_idref="BR08">
  <Loan_History Loan_Date="Mar-3-2002" Loan_History_idref="BW09"/>
</Borrow>

<Borrower Borrower_No="B22" Borrower_Name="Riveredge" Borrower_idref="BR08"/>

<Record_Folder Folder_No="F_24" Location="New Territories" HKID="E3766849" Record_Folder_idref="RF04">
  <Medical_Record Medical_Rec_No="M_333333" Create_Date="Mar-1-2001" Sub_Type="A">
    <AE_Record AE_No="AE_3"/>
  </Medical_Record>
</Record_Folder>

<Borrow Borrower_No="B22" Folder_No="F_24" Borrow_id="BW10" Record_Folder_idref="RF04" Borrower_idref="BR08">
  <Loan_History Loan_Date="Apr-3-2002" Loan_History_idref="BW10"/>
</Borrow>

<Borrower Borrower_No="B22" Borrower_Name="Riveredge" Borrower_idref="BR08"/>

<Record_Folder Folder_No="F_23" Location="New Territories" HKID="E6077888" Record_Folder_idref="RF03">
  <Medical_Record Medical_Rec_No="M_382001" Create_Date="Feb-22-2001" Sub_Type="O">
    <Outpatient_Record Outpatient_No="O_53" Specialty="Therapy"/>
  </Medical_Record>
</Record_Folder>

<Borrow Borrower_No="B25" Folder_No="F_23" Borrow_id="BW11" Record_Folder_idref="RF03" Borrower_idref="BR09">
  <Loan_History Loan_Date="Mar-03-2002" Loan_History_idref="BW11"/>
</Borrow>

<Borrower Borrower_No="B25" Borrower_Name="Golden Park" Borrower_id="BR09"/>
</Borrow_info>