XML Structures for Relational Data

Wenyue Du Mong Li Lee Tok Wang Ling
School of Computing
National University of Singapore
Singapore 117543
{duwenyue, leeml, lingtw}@comp.nus.edu.sg

Abstract

XML is increasingly being adopted for information publishing on the World Wide Web. However, the underlying data is often stored in the relational databases. Some mechanism is needed to convert the relational data into XML data. In this work, we employ a semantically rich semistructured data model, the Object-Relationship-Attribute model for semistructured data, as a middleware to support the schema conversion from semantically enriched relational schema to XML Schema. This approach allows us to handle the translation of a set of related relations and to distinguish attributes of relationship types from attributes of object classes, multi-valued attributes, and different types of relationships such as binary, n-ary, recursive and ISA. The resulting XML structures are able to reflect the inherent semantics and implicit structure in the underlying relational database. We also show that the appropriate use of references is able to avoid unnecessary redundancy and the proliferation of disconnected XML elements.

1. Introduction

XML is emerging as a standard for information publishing on the World Wide Web. However, the underlying data is often stored in traditional relational databases. Some mechanism is needed to convert the relational data into XML data. We can classify existing approaches to publish XML data from relational databases as follows:

1. Customized translation of relational data to a "pre-defined" schema for XML data. For instance, a new language *RXL* to specify XML views of the relational data is proposed in [6] and extended *Nested SQL* statements are introduced in [12] to specify XML element construction.
2. No "pre-defined" schema information is required. That is, *default* XML views are produced according to the structures of the relations in [1,13,15].

One of the major challenges in both the approaches is to find an effective way to generate an XML structure that is able to describe the semantics and structure in the underlying relational database. XML consists of nested element structures and the relationships of elements are modeled directly by hierarchies and references. In contrast, relational data is flat and normalized. As a consequence, the translation from relational data to XML is often not intuitional but rather complex. The first approach utilizes a significant amount of customized code to construct the XML structure, which is typically subjective and inaccurate. The transformation techniques employed in the second approach currently lack a detailed analysis of the relational schema and focus on single relation conversions. As a result, the XML data is either flattened into tuples containing many redundant elements, or has many disconnected XML elements.

In this paper we develop a methodology which employs the semantically rich Object-Relationship-Attribute model for semistructured data (ORA-SS) [5] in the translation process. ORA-SS has characteristics which are very similar to XML: self-describing, deeply nested or even cyclic, and irregular. At the same time, ORA-SS models a rich variety of semantic constraints in the underlying relational database, and represent the implicit structures of relational data using hierarchy and referencing. In our proposed relational to XML Schema translation, we want to satisfy the following requirements:

1. Generate an XML structure that is able to describe the semantics and structure in the underlying relational database.
2. Allow the translation of a set of related relations instead of simple single relation/relationship conversions.
3. Obtain properly structured XML data without unnecessary redundancies and proliferation of disconnected XML elements.

Figure 1 shows the steps in our proposed translation:

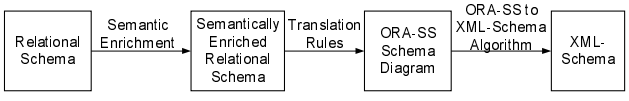

Figure 1. Relational-to-XML Translation

In the semantic enrichment of a relational schema, we will identify the following inherent semantics and implicit structure in the relational schema:

1. Object relations that represent regular and weak entity types.
2. Relationship relations that represent various relationship types, such as binary, n-ary, recursive and ISA (inheritance) relationship types.
3. Fragments of object relations or relationship relations that represent single-valued and multivalued attributes of entity types or relationship types.
4. Cardinality constraints

The semantic information is then represented explicitly in an ORA-SS schema diagram from which an XML Schema is subsequently derived. We will present a set of *translation rules* to translate a semantically enriched relational schema to an ORA-SS schema diagram, and an algorithm to generate an XML Schema from an ORA-SS schema diagram.

The rest of the paper is organized as follows. Section 2 reviews the concept of semantic dependencies and illustrates how it can be used to provide a more accurate analysis of the relational schema. Section 3 presents the rules to translate a semantically enriched relational schema to an ORA-SS diagram. An algorithm to generate an XML Schema from an ORA-SS diagram is also given. Section 4 gives a discussion of related work and we conclude in Section 5.

2. Semantic Enrichment of Relational Schema

The semantic enrichment of relational schemas has been extensively studied in [3, 7, 9, 11]. Functional dependencies and inclusion dependencies have traditionally been used to aid the translation of relational database into semantic data models such as the Entity-Relationship model [4] and the object-oriented model [2]. However, functional dependencies and inclusion dependencies are basically constraints to enforce the integrity of a database. [9] introduces the concept of *semantic dependencies* to represent the relationship between two sets of attributes at a semantic level. We will use the relational schema of a *university* database shown in Figure 2 to review and illustrate the main concepts.

1. An *entity key* denotes the identifying attribute(s) of real world entities. The notion of an entity key is different from the traditional concept of a key.
   For instance, in the all-key *HOBBIES* relation in Figure 2, the entity key is *S#* because it identifies the entity type *STUDENT* uniquely while *HOBBY* is just an attribute of *STUDENT*.
2. An attribute *A,* which is not a part of any entity key, is said to be *semantically dependent* on a set of entity keys if the value of A needs to be updated whenever the value of some entity key in the set changes.
   Consider the *STUDENT* relation in Figure 2. Suppose *REGISTRATIONDATE* is the date when a student registers at a department. *REGISTRATIONDATE* is semantically dependent on *{S#, D#}*, denoted by *{S#,D#}* $\xrightarrow{SD}$ *REGISTRATIONDATE*. This indicates that *REGISTRATIONDATE* is meaningful only when associated with *S#* and *D#* together. Note that *REGISTRATIONDATE* is functionally dependent on only *S#* because a student can only register at only one department.
3. Two entity keys are *semantically equivalent* if they both identify the same entity type.
4. A set of entity keys is a *semantic key K* of a relation if any semantic dependency *K'* $\xrightarrow{SD}$ *A* in *R* implies *K* is semantically equivalent to *K'*.
   For instance, entity keys *CODE* is the semantic key of *Course*, and *{CODE, S#}* is the semantic key of relation *C_S_1*. In Figure 2, all the semantic keys in the *university* database are indicated in bold.

COURSE(**CODE**, TITLE) The courses held in the university DEPT(**D#,** DNAME) The department of the university TUTORIAL(**T#,** TUTORIALTITLE) The tutorials of the courses NOTES(**NOTE-ID**, LECTURER) The notes for a course, provided by one lecturer HOBBIES(**S#,** HOBBY) The hobbies of a student	**Group 1**
STUDENT (S#, SNAME, REGISTRATIONDATE,D#) The students registering to a department.	**Group 2**
COURSE_NOTES (**NOTE-ID**, **CODE**) Defines a 1-m relationship between course and notes C_S_1(**CODE, S#,** GRADE) Defines a m-n relationship between course and student ATTEND(**CODE, T#, S#**) Defines a ternary relationship among course, student and tutorial COURSEMEETING (**CODE, S#,** MEETINGHISTORY) Records meeting histories of courses and students.	**Group 3**

Figure 2. Relational schema of a university database

The above concepts are useful in clustering the relations and attributes in a relational schema. Relations are classified into the following three types:

- *Object relation* whose semantic key consists of only one entity key, or more than one entity keys which are semantically equivalent.

- *Relationship relation* whose semantic key consists of more than one entity keys which are not semantically equivalent.
- *Mix-type relation* which does not have any semantic key. This type of relations will be subsequently split into object relations and relationship relations.

Example 1. Consider the *university* database in Figure 2. The relations are classified into object relations, mix-type relations and relationship relations as indicated by Group 1, Group 2 and Group 3 respectively. *STUDENT* relation under Group 2 is a mix-type relation because it has two semantic dependencies $S\# \xrightarrow{SD} SNAME$ and $\{S\#, D\#\} \xrightarrow{SD} REGISTRATIONDATE$, but *S#* and *{S#, D#}* are not semantically equivalent. We split relation *STUDENT* into an object relation and a relationship relation as follows:

STUDENT (S#, SNAME)
STUDENTDEPT (S#, D#, REGISTRATIONDATE)

Attributes are classified into *object attributes* and *relationship attributes*:

- If an attribute is semantically dependent on exactly one entity key or more than one entity keys which are semantically equivalent, then it is an object attribute.
- If an attribute is semantically dependent on more than one entity keys which are not semantically equivalent, then it is a relationship attribute.

REGISTRATIONDATE, *GRADE* and *MEETING-HISTORY* are relationship attributes in the *university* database (See Figure 2).

By using semantic dependencies together with functional dependencies and inclusion dependences, we can identify relationship relations that represent binary, ternary, recursive[1], or ISA relationship type[2], and object relations that represent weak entity type[3]. From the multivalued dependencies, we can identify that the *HOBBIES* relation is a *fragment*[4] of the object relation *STUDENT*, and *COURSEMEETING* is a fragment of the relationship relation *C_S_1*. Furthermore, we can also establish the cardinalities of relationship types. This is important to generate an XML Schema correctly. The different possible cardinalities include 1-1, 1-m, m-1 and m-n. For the rest of the paper, we shall assume that the relations and attributes in a relational schema have been clustered, and the various relationship types and cardinality constraints have been identified as shown in Figure 2.

[1] A recursive relationship type is one in which an entity type participates more than once, assuming a different role upon each entry type into the relationship type.
[2] An ISA relationship type indicates that a lower-level entity type is formed by taking a subset of a higher-level entity type.
[3] The existence of a weak entity type entity depends on the existence of an associated regular entity type entity.
[4] Fragment relations are caused by the existence of multi-valued attribute of the entity type.

3. Relational to XML Translation

3.1 ORA-SS Model

In our study, we found that the quality of the resulting XML Schema depends not only on the transformation methodology, but also on the expressiveness of the chosen semistructured data model. We adopt ORA-SS because it is a semantically richer data model that has been proposed for semistructured data compared to the existing models such as OEM [10], XOM [17]. We will briefly review the ORA-SS model in this section.

ORA-SS distinguishes between object classes, relationship types and attributes. In an ORA-SS schema diagram, object classes are denoted by labelled rectangles and relationship types are denoted by directional labelled edges. The direction of the edge is from the *parent* object class to the *child* object class. The label indicates the information of relationship *name*, *degree*, *cardinality* constraint on the parent and child object class. Attributes are denoted by labelled circles and keys are indicated by filled circles. ORA-SS not only reflects the hierarchy structure of semistructured data, but also provides references to indicate that the *referenced object* class is not materialized in a nesting relationship within its parent. References are denoted by the dashed arrows from a *referencing* object class to a referenced object class.

Figure 3 shows one of the possible ORA-SS schema diagrams that models the *university* database. *COURSE, STUDENT* and *DEPT* are *root* object classes. *COURSE* has a child object class *NOTES*. The label (2,1:n,1:1) on the edge between *COURSE* and *NOTES* indicates a binary relationship type between *COURSE* and *NOTES* (denoted by 2). There can be one or many *NOTES* for each *COURSE* (denoted by 1:n). A *NOTES* is used by only one *COURSE* (denoted by 1:1). Note that such cardinality constraints can be obtained from the semantically enriched schema.

By employing the ORA-SS schema diagram as a middleware in the relational to XML translation, we can separate the task of designing XML Schema from the detailed analysis of semantics and structures of underlying legacy data. Furthermore, ORA-SS offers a visually effective way of designing and maintaining XML Schema. We note that traditional semantic data models such as the Entity-Relationship model cannot support XML naturally and fully. For example, reference is an important concept in XML schema. This feature can be explicitly represented using an ORA-SS schema but not the ER model.

3.2 From Semantically Enriched Relational schema to ORA-SS

We will now present three sets of rules for translating a semantically enriched relational schema to an ORA-SS schema diagram.

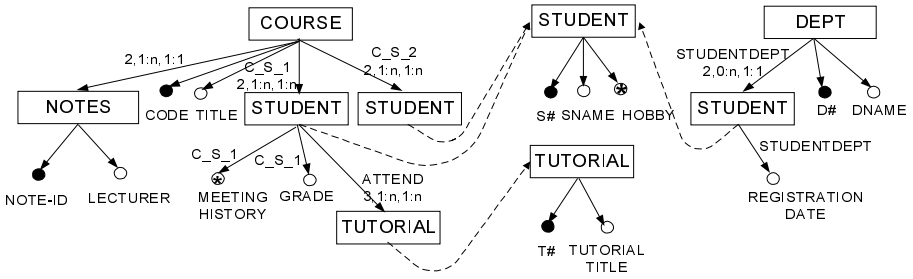

Figure 3. An ORA-SS Schema Diagram

1. Object relation rules that translate object relations into ORA-SS object classes.
2. Relationship relation rules that translate relationship relations into ORA-SS relationship types represented as hierarchies and references.
3. Combination rules that are applied to the result obtained from the application of object and relationship rules to generate the final translation.

We note that the translation of a relational schema to an ORA-SS schema diagram differs from the translation of a relational schema to a hierarchical model. This is because there are distinct differences between ORA-SS/XML and traditional hierarchical databases (e.g., IMS system). For example, we can have cycles or self-referencing in XML but not in hierarchical database (e.g., IMS system). Moreover, virtual pointers in hierarchical databases cannot have further structures (e.g. attributes and child record types) as reference elements in XML.

3.2.1 Object Relation Translation Rules

Rule O1: Regular Object Relation Rule. Create an ORA-SS object class O for each regular object relation R. The connecting structure of these object classes depends upon the relationship types among them. Each attribute of R is mapped to an ORA-SS object attribute of O. The primary key of R becomes the key of O. Note that attributes of a regular object relation are mapped into *single*-valued attributes of O. □

Rule O2: Fragment of Object Relation Rule. Each fragment R_f of an object relation R is mapped into an ORA-SS attribute A of an object class O_R, where O_R is the ORA-SS object class corresponding to R. The cardinality of A is determined by the cardinality of R_f. Table 1 shows the mapping rules. □

Example 2. *HOBBIES* (R_f) is a fragment of the object relation *STUDENT* (R). Attribute *HOBBY* (K_1) in the relation *HOBBIES* is mapped to a simple multivalued attribute *HOBBY* (A) (labelled by "*") of *STUDENT* (O_R). This mapping is shown in Figure 4.

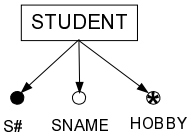

Figure 4. A fragment relation is mapped to a multivalued object attribute

Rule O3: Weak Entity Type Rule. Each object relation R_B which represents a weak entity, is mapped to a child object class O_B of O_A which represents the associated regular object relation R_A. Note that R_B is viewed as a composite multivalued attribute of O_A if R_B does not contain non-key attribute. Each attribute of R_B is mapped to an object attribute of O_B, except the entity key E_B (which is a part of the primary key of R_B) with the inclusion dependency: $R_B[E_B] \subseteq R_A[K_A]$, where K_A is the semantic key of R_A. □

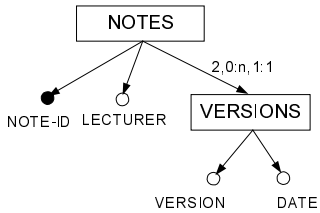

Figure 5. An object relation representing weak entity type is mapped to a child object class

Example 3. Suppose object relation *VERSIONS (NOTE-ID, VERSION, DATE)* defines the versions of a lecture notes. Assume that we identify *VERSIONS* (R_B) represents a weak entity type entity of *NOTES*, then it is mapped to a child object class *VERSIONS* (O_B) of the object class *NOTES* (O_A). The entity key *NOTE-ID* will *not* be mapped as the key of *VERSIONS*. This indicates that *VERSIONS* must be associated with the object class *NOTES*. Figure 5 shows the mapping.

Definition of R_f	Mapping Rules		
	n>1	n=1	Cardinality of O_R - $\{A_1 \ldots A_n\}$
R_f ($\underline{K}$, $\underline{A_1 \ldots A_n}$) K $\xrightarrow{SD}$ $\{A_1 \ldots A_n\}$	$\{A_1 \ldots A_n\}$ is mapped to a composite multivalued attribute	A_1 is mapped to a simple multivalued attribute	m-n
R_f (K, $\underline{A_1 \ldots A_n}$) K $\xrightarrow{SD}$ $\{A_1 \ldots A_n\}$			1-m
R_f ($\underline{K}$, $\underline{A_1 \ldots A_n}$) K $\xrightarrow{SD}$ $\{A_1 \ldots A_n\}$	$\{A_1 \ldots A_n\}$ is mapped to a composite single-valued attribute	A_1 can be viewed as an attribute	1-1
R_f ($\underline{K}$, $A_1 \ldots A_n$) K $\xrightarrow{SD}$ $\{A_1 \ldots A_n\}$			m-1

* $R_f[K] \subseteq R[K_R]$, where K_R is the semantic key of relation R.

Table 1. Mapping Rules for the Cardinality of an Attribute

3.2.2 Relationship Relation Translation Rules

Theoretically, ORA-SS allows the cardinality constraint on the child object class to be "zero". However, in an XML document, each child element must be associated with a parent element. Here, we enforce the cardinality constraint on the child object classes to be "one" or more.

In the following rules, we assume R_{AB} is a binary relationship relation where its semantic key consists of two entity keys of two entity types *A* and *B*. *A* and *B* are represented as two object relations R_A and R_B respectively.

Rule R1: 1-m Relationship Rule. Let R_{AB} represent a 1-m relationship type (say *R*), the cardinality of entity type *A* is "one", and the cardinality of entity type *B* is "many".

Case 1: If all the entities of *B* participate in *R*, then R_A is mapped to a parent object class O_A and R_B is mapped to a child of O_A.

Case 2: If all the entities of *A* participate in *R*, and R_B has been mapped as a child of another object class or not all the entities of *B* participate in the relationship, then R_B is mapped to a parent object class O_B and R_A is mapped to a child of O_B.

Case 3: If there exist entities of either *A* or *B* not participating in *R*, then R_A and R_B are mapped to O_A and O_B respectively. O_A and O_B is connected using reference.

Each relationship attribute is mapped to an attribute of an ORA-SS relationship type. If the associated relationship is represented using references, then we attach the attribute to the referencing object class. Otherwise, we attach it to the child object class. □

Example 4. Suppose *STU_ADVISOR*(*S#, STAFF#)* represents a 1-m relationship type involving object relations *ADVISOR(STAFF#, POSITION)* and *STUDENT*. The cardinality of *ADVISOR* is "one" and that of *STUDENT* is "many". Note that *STUDENT* cannot be mapped as a child of *ADVISOR* if not all students are assigned to an advisor. According to Case 2 in Rule R1, *ADVISOR* is a child object class of *STUDENT* if every advisor must advise one or more students. This mapping is shown in Figure 6.

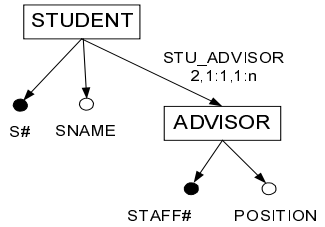

Figure 6. A 1-m relationship type is mapped to a hierarchical structure

Rule R2: m-n Relationship Rule. Let R_{AB} represent a m-n relationship type, it is translated as follows: The referencing object is set below one object class (say O_A) to connect the other (say O_B) (the referencing direction may be decided by applications). Particularly, referencing object is used on both sides in order to describe the symmetric relationship. The relationship attributes are attached to the referencing object(s). □

Example 5. Consider the m-n relationship relation *C_S_1* with object relations *COURSE* (R_A) and *STUDENT* (R_B). Existing works have handled m-n relationship types by creating a new object class to aggregate the references which connect the participating object classes. Figure 7 shows an ORA-SS schema diagram that aggregates referencing attributes for the *C_S_1*. This creates too many references and cause poor query response time while avoiding data redundancies. In additional, such a flat structure is not suitable to represent the relationship at the semantic level. In contrast, Figure 8 shows one of possible ORA-SS schema diagrams produced according to Rule R2. We conducted some experiments to compare the performance of the various ways to map an m-n relationship. Our experiments proved that the performance of direct referencing is much better than introducing a new structure to aggregate the references as we try to navigate from one element (say *COURSE*) to others (say *STUDENT*) through references. Besides, such hierarchical structure is semantically richer.

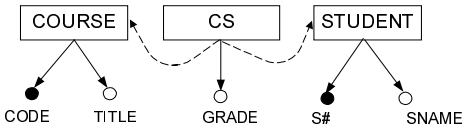

Figure 7. A m-n relationship is mapped to a new object class

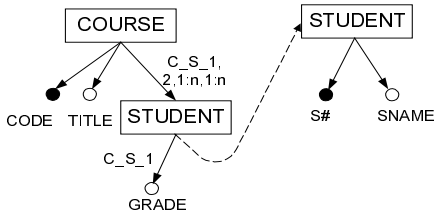

Figure 8. A m-n relationship is mapped to a reference

Rule R3: Recursive Relationship Rule. Recursive relationship type is translated as follows: The participating object relation is mapped to an ORA-SS object class (say O), and the referencing object is set below the object class to connect O. In order to describe the symmetric relationship, two collections of referencing attributes need to be used to reference to O. One collection holds those objects to which it contributes, and the other holds those objects that it comprises. □

Rule R4: ISA Relationship Rule. ISA relationship type is represented as the hierarchy structure. If B ISA A, then R_B is mapped to a child object class (O_B) of O_A. Note that the entity key E_B of R_B with the following inclusion dependencies: $R_B[E_B] \subseteq R_A[K]$, where K is the semantic key of R_A, need to be contained in O_B. □

Example 6. Suppose we have an object relation *PERSON(SSNO, GENDER, RACE),* where social security No. (SSNO) also appears in *STUDENT* relation as a candidate key. We can identify that *STUDENT* ISA *PERSON* by the enrichment algorithm [9]. Figure 9 shows the mapping of such an ISA relationship type. The entity key *SSNO* (E_B) is mapped to the candidate key attribute of both *PERSON* (O_A) and *STUDENT* (O_B) because it is the identifying attribute of *PERSON* as well as *STUDENT*.

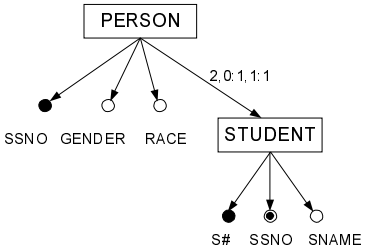

Figure 9. An ISA relationship type is mapped to a hierarchical structure

Rule R5: n-ary Relationship Rule. Object relations such as R_1, R_2, ..., R_n participating in n-ary relationship type can be translated as follows: Let object relations R_1, R_2, ..., R_n be mapped to ORA-SS objects O_1, O_2, ..., O_n respectively. We choose a path and create the referencing objects sequentially (the path may be decided by applications). Without loss of generality, create a referencing element (say O_2^r) below O_1 to connect with O_2, and then create a referencing object O_3^r below O_2^r to connect with O_3, and so forth until all the participating objects are connected. Particularly, the level of each referencing object is determined by the aggregations, where aggregations are a means of enforcing inclusion dependencies in a database, if the aggregations are available. □

Example 7. Suppose the object relations *COURSE*, *STUDENT* and *TUTORIAL* participate in a ternary relationship *ATTEND*. If we have the following inclusion dependencies: $ATTEND[CODE,S\#] \subseteq C\_S\_1[CODE,S\#]$, then we will create the referencing object class *TURORIAL* at the lowest level to enforce referential integrity as shown in Figure 3.

Rule R6: Fragment Relationship Relation Rule. Each fragment of relationship relation is mapped to a composite and/or multivalued attribute in the same way as the translation of the fragment of object relation. The edge connecting to such attributes is tagged with the relationship name in order to show which relationship it belongs to. □

Example 8. Attribute *MEETTINGHISTORY* in the relation *COURSEMEETING*, which is a fragment of *CS* ($COUSEMEETING[CODE, S\#] \subseteq C\_S\_1[CODE,S\#]$), is mapped to a simple multivalued attribute *MEETING-HISORY* below the referencing object *STUDENT* (ref. to Figure 8). Figure 10 shows the mapping.

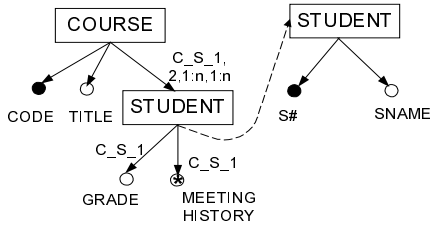

Figure 10. A fragment relation is mapped to a multivalued attribute of a relationship type

3.2.3 Combination Rules

Consider the case where an object relation R is a candidate child object class for more than one relationship type. For example, we can have *STUDENT* and *PERSON* participate in the ISA relationship type. At the same time,

Translation of	Priority	Reasons
Fragments of Object class	1st	These fragments actually represent attributes of an object.
ISA, weak type and recursive relationship type and their fragments	2nd	There exist high semantic cohesion among these participating objects.
1-1 and 1-m relationship type and their fragments	3rd	These relationship types are potentially represented as hierarchy structure.
m-1 relationship type and their fragments	4th	M-1 relationship is potential hierarchy structures. Note that we usually view it as 1-m relationship in order to reduce redundancies caused by nesting.
m-n, n-ary relationship type and their fragments	5th	

Table 2. Priority rules

STUDENT and *DEPT* participate in the 1-m relationship type *STUDENTDEPT*. We observe that *STUDENT* is a potential child object class of either *PERSON* or *DEPT*, and we need to decide which one should be its parent object class. Note that if we use references to connect *STUDENT* with *DEPT* and *PERSON,* it will induce many disconnected XML elements and cause poor query response time. With this in mind, we use the notion of *cohesion* proposed in [14] to represent the strength of the relationship among entities. This notion has been applied in clustering technique to generate some desired level of abstraction. The cohesion concept helps us to decide which object class can be the parent in the case where one object relation *O* potentially has "multiple parent". Here, we choose the one which has stronger cohesion with *O* as the parent object class. We therefore prioritize the translations of different relationship types to ensure the parent object classes derived always have the strongest cohesion with *O*.

Rule C1. Translations are prioritized according to *cohesion* between object classes. Translations are produced sequentially according to their priorities. The translation with the lowest priority will be carried out last. Table 2 shows the priorities of translations. □

According to Table 2, *STUDENT* is translated to a child object class of *PERSON* first, and then *DEPT* is placed below *STUDENT* according to Case 2 in Rule R1. The prioritized translations ensure that *PERSON* can be mapped as the parent of *STUDENT*. Note that user's input is needed in the case when it is not clear which relationship has higher cohesion.

Rule C2. If an object class *O* participates into more than one relationship type, then it should not be mapped to a child object class of either relationship type if the mapping induces *O* to be referenced by other object classes. □

Example 9. Consider the object relation *STUDENT (R),* which is involved in the m-n relationship *C_S_1* and the 1-m relationship type *STUDENTDEPT*. *REGISTRATIONDATE* and *GRADE* is relationship attribute of *STUDENTDEPT* and *C_S_1* respectively. Rule C1 will generate the ORA-SS diagram in Figure 11. Users can travel from *COURSE* elements to *STUDENT* elements but not vice versa. However, a better ORA-SS diagram can be obtained in Figure 12 if we apply Rule C2. *STUDENT*, *DEPT* and *COURSE* will be translated into root object classes. *STUDENT* is connected to *DEPT* and *COURSE* using references. The benefits are (1) such a structure ensures that the referenced elements will not contain repeated instances, and (2) attributes belonging to different relationships can be distinguished separated from each other as well as other object attributes.

Example 10. The ORA-SS diagram shown in Figure 3 can be derived as follows. We assume that the essential information are *DEPT* and *COURSE*, and translate them into root object classes. Object relation *HOBBIES* is identified as a fragment of *STUDENT* and therefore mapped as a simple multivalued object attribute according to Rule O2. *NOTES* is mapped to a child object class of *COURSE* according to Case 1 in Rule R1. According to Rule C2, *STUDENT* is translated to a root object class that *COURSE* and *DEPT* connect with it using references. *COURSEMEETING* is identified as a fragment of *C_S_1* and mapped as a simple multivalued relationship attribute according to Rule O2. *COURSE*, *STUDENT* and *TUTORIAL* participate in a ternary relationship type *ATTEND*, so they are translated according to Rule R5. Note that there exist two different m-n binary relationship types, *C_S_1* and *C_S_2* between *COURSE* and *STUDENT*. In such case that a ternary relationship type (i.e., *ATTEND*) involves only one of the binary relationship types (i.e., *C_S_1*) between two object classes, we need to explicitly separate the two referencing objects in order to distinguish which binary relationship type it involves. The whole translation order is in accordance to Rule C1.

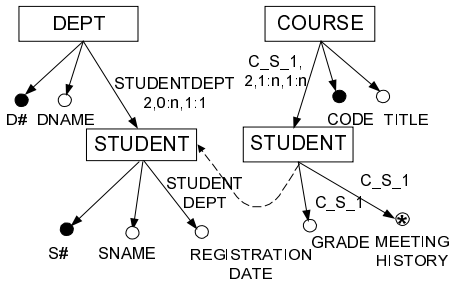

Figure 11. An undesirable structure

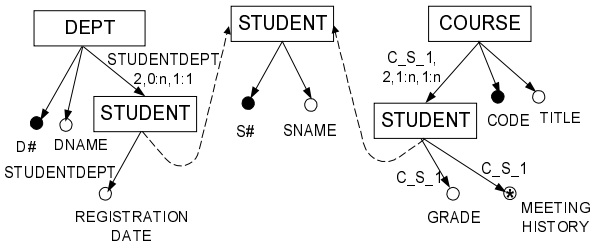

Figure 12. A preferred structure

3.3 From ORA-SS to XML Schema

When the ORA-SS schema diagram of a relational database is obtained, an XML Schema can now be derived relatively easily because of the additional semantics captured in ORA-SS. In addition, XML features such as the concept of *reference* has also been taken into consideration in the process of deriving ORA-SS diagram (recall the relationship translation rules). The following algorithm gives the translation of an ORA-SS schema diagram to an XML schema.

Algorithm ORASS-to-XML

Step 1 Declare an XML element for whole schema and create a complex type.

Step 2 For each object o in ORA-SS schema diagram Do

Case 1: o is a root object class

Declare an element directly below R_o and create the corresponding complex type.

Case 2: o is a referencing object

Declare referencing attribute(s) K_r, or a sub-element which contains such attribute(s) K_r if there exists relationship attribute(s) below *o*. The sub-element is named to the relationship name.

Case 3: o is a child object class

Declare a sub-element of the corresponding element, and create the corresponding complex type.

- Set the content of the generated XML elements to *EMPTY* because they are either references, or indirectly translated from relations, which have no value.
- Set the cardinality constraints[5] of each element according to corresponding cardinality label in ORA-SS diagram.

Step 3 Declare a XML attribute (or element) for each attribute of each ORA-SS object, and assign proper XML type[6] and cardinality[7].

Step 4 Add keys and key references in XML Schema for the keys of ORA-SS object.

Note that in Step 3 of the algorithm, single-valued, multivalued and composite attributes can be represented as various XML structures. Usually it is hard to tell which structure is better. For example, a multivalued attribute can be declared as an XML attribute and typed as *NMTOKENS*, which is a list type. Alternatively, it can be declared as a sub-element which consists of one or more occurrences. For our example, we declare multivalued attribute as sub-element while single-valued attribute as XML attribute. However, the relationship attribute, which is attached in a child object class (say *O*), is declared as a sub-element in order to be distinguished from the object attributes of *O*. It is worthwhile to point out that, if an ORA-SS edge is tagged with several labels, it indicates that there exist several relationship types between the two object classes. In this case, we need to materialize the component object class in each relationship it participates. In order to reduce redundancies, we may materialize it in the one with smaller degree first, and then use *keyref* to refer it from other relationship types. Appendix A. shows the XML Schema derived from Figure 3.

4. Related Works and Discussions

Existing approaches to the relational to XML translation do not regard whether the resulting XML structure correctly describe the semantics and structure in the underlying relational database. The works in [1, 13, 15] basically focus on single relation conversions. In order to handle a set of related relations, the relations are first denormalized to one single relation. Unfortunately, this will lead to a lot of redundancies in the resulting XML instances. In addition, the resulting schema is semantically weak. For instance, suppose some user requires all the information of the relations *COURSE* and *STUDENT*. The XML structure produced by [1, 13, 15] will have the following *flat* structure:

RESULTS(CODE, TITLE, S#, SNAME, GRADE)

---

[5] The default cardinality defined in XML schema for elements is exactly one, denoted by minOccurs = "1" and maxOccurs = "1"

[6] XML schema provides rich types which can support most types in the underlying database.

[7] The cardinality defined in XML schema for attributes is zero or one (denoted by use="optional"), or exactly one (use="required"). The default value is *optional*.

The authors in [8] propose a *Nesting-based* algorithm to convert a single relation to a DTD. However, this algorithm is applied on extracted data sets. Different data sets extracted will lead to different structures which do not reflect the semantics in the underlying database. For instance, suppose *COURSE* and *STUDENT* participate in a m-n relationship. However, if a particular extracted data set shows that a student only takes one course, then the XML structure derived will depict a 1-m relationship and not a m-n relationship.

A *naive* approach to handle a set of related relations will be to translate each relation to an XML element. The various elements are then connected by referencing elements or attributes in order to model the foreign key constraints. One of the major problems of this approach is the proliferation of references that will lead to performance degradation. Furthermore, the schema of XML data obtained is flat.

[16] develops a method to generate a hierarchical DTD for XML data from a relational schema. First, one or more relations are chosen as the XML root elements, and then each sub-element is progressively defined by travelling across relations via the foreign key constraints. While this translation is intuitive and effective, problems still arise. For example, if we define *STUDENT* as a sub-element of *ADVISOR*, then we cannot represent those students who have not been assigned advisor yet.

In contrast, our proposed relational to XML translation method provides for the translation of a set of related relations and distinguishes attributes of relationship types from attributes of object classes, multivalued attributes, different types of relationships such as binary, n-ary, recursive and ISA. The structure of the XML data obtained is able to reflect the inherent semantics and implicit structure in the underlying relational database without unnecessary redundancy and proliferation of disconnected XML elements.

5. Conclusion

In this paper, we have proposed an alternative practical methodology for publishing XML data from relational databases. We have shown the importance of proper analysis of semantics in relational schema. The design of a semantically sound XML structure for relational data is a complicated task that needs users' input. With user input, we can provide an XML schema that is closer to the user expectation, and that preserves the inherent semantics and implicit structure in relational schema. For future work, we would like to examine how data mining techniques can be used to mine the semantic information in XML schemas.

References

[1] S. Banerjee, V. Krishnamurthy, M. Krishnaprasad, R. Murthy, "Oracle 8i – The XML Enabled Data Management System", *Proc. 16th Int'l Conf. on Data Engineering,* 2000

[2] E. Bertino, L. Martino, "Object-oriented Database Management Systems: Concepts and Issues," *IEEE Computer 24:4,* 1991

[3] M. Castellanous, F. Saltor, "Semantic Enrichment of Database Schema: An Object-Oriented Approach", *Proc. First Int'l Workshop on Interoperability in Multidatabases,* pages 71-78, 1991

[4] P.P Chen, "The Entity-Relationshiop Approach to Logical Database Design", *Q.E.D. Information Sciences,* 1977.

[5] G. Dobbie, X.Y. Wu, T.W. Ling, M.L. Lee, "ORA-SS: Object-Relationship-Attribute Model for Semistructured Data", *TR 21/00, National Univ. of Singapore*, 2001

[6] M. Fernandez, W.C. Tan, D. Suciu, "SilkRoute: Trading between Relations and XML," *Proc. 9th Int'l WWW Conf.,* 2000

[7] J-L Hainaut, M. Chandelon, et al., "Transformational Techniques for Database Reverse Engineering", *Proc. 12th Int'l Conf. on ER Approach*, 1993

[8] D.W. Lee, M. Mani, F. Chiu, W.W Chu, "Nesting-based Relational-to-XML Schema Translation", *Proc. 4th In'l Workshop on the Web and Databases*, 2001

[9] T.W. Ling, M.L. Lee, "Relational to Entity-Relationship Schema Translation Using Semantic and Inclusion Dependencies", *In Journal of Integrated Computer-Aided Engineering,* pages 125-145, 1995

[10] J. McHugh, S. Abiteboul et al., "Lore: A Database Management System for Semistructured Data", *SIGMOD Record,* 26(3):54-66, 1997

[11] R. Missaoui, J.M. Gagnon, R. Godin, "Mapping an Extended Entity-Relationship into A Schema of Complex Objects", *Proc. Int'l Conference on Object Oriented and Entity Relationship Modeling,* 1995

[12] J. Shanmugasundaram et al, "Efficiently Publishing Relational Data as XML Documents", *Proc. 26th In'l Conf. on Very Large Databases,* 2000

[13] SYBASE, "Using XML with the Sybase Adaptive Server SQL Databases, A Technical Whitepaper", http://www.sybase.com 2000

[14] T. Teorey et al., "ER Model Clustering as an Aid for User Communication and Documentation in Database Design", *CACM 32(8),* pages975-987, 1989

[15] V. Turau, "Making Legacy Data Accessible for XML Applications", http://www.informatik.fhiesbaden.de/~turau/veroeff.html1999

[16] K. Williams, et al., "XML Structures for Existing Databases", http://www-106.ibm.com/developerworks/library/x-struct/ January 2001.

[17] D. Zhang, Y.S. Dong, "A Data Model and Algebra for The Web", *Proc. 10th Int'l Workshop on Database and Expert Systems Applications*, pages711 –714, 1999

Appendix A. XML Schema derived from Figure 3

```
<!— declare an element for whole schema -->
<  element name="UNIVERSITY"
            type=  "UNIVERSITY_TYPE" />

<  complexType name="UNIVERSITY_TYPE"
                content="empty" >
   <  element name="STUDENT" type="STUDENT_TYPE"
            maxOccurs=" unbounded"/>
   <  element name="DEPT" type="DEPT_TYPE"
            maxOccurs=" unbounded"/>
   <  element name="COURSE" type="COURSE_TYPE"
            maxOccurs=" unbounded"/>
   <  element name="TUTORIAL" type="TUTORIAL_TYPE"
            maxOccurs=" unbounded"/>
</ complexType >

<!— define a complex type for each sub-element of
UNIVERSITY, and declare its sub-elements progressively -->

<  complexType name="DEPT_TYPE" content="empty" >
   <  attribute name="D#" type="string" use="required"/>
   <  attribute name="DNAME" type="string" use="required"/>
   <  element name="STUDENT" minOccurs="0">
      <  complexType content="empty">
         <  attribute name="STU_REF" type="string" />
         <  attribute name="REGISTRATIONDATE"
                  type="date" >
      </ complexType >
   </ element >
</ complexType >

<  complexType name="COURSE_TYPE" >
   <  attribute name="CODE" type="string" use="required"/>
   <  attribute name="TITLE" type="string" use="required"/>
   <  element name="C_S_2" minOccurs="0" >
      <  complexType content="empty">
         <  attribute name="STU_REF" type="string" />
      </ complexType >
   </ element >
   <  element name="C_S_1" minOccurs="0" >
      <  complexType content="empty">
         <  attribute name="GRADE" type="number" />
         <  element name="MEETINGHISTORY"type="string"
                  minOccurs="0" maxOccurs="unbounded"  />
         <  attribute name="STU_REF" type="string" />
         <  attribute name="TUTORIAL_REF" type="string" />
      </ complexType >
   </ element >
   <  element name="NOTES" >
      <  complexType content="empty" >
         <  attribute name="NOTE-ID" type="string"
                  use="required"/>
         <  attribute name="LECTURER" type="string" />
      </ complexType >
   </ element >
</ complexType >

<  complexType name="STUDENT_TYPE" content="empty" >
   <  attribute name="S#" type="string" use="required"/>
   <  attribute name="SNAME" type="string" use="required"/>
<!— declare an element for multivalued  ORA-SS attribute
"HOBBY"  -->
   <  element name="HOBBY" type="string"
            minOccurs="0" maxOccurs="unbounded" />
</ complexType >

<  complexType name="TUTORIAL_TYPE"
                content="empty" >
   <  attribute name="T#" type="string" use="required"/>
   <  attribute name="TUTORIAL_TITLE" type="string"/>
</ complexType >

<!— define keys and keyref  constraints. -->

< key name="STUDENT_KEY" >
      < selector >UNIVERSITY/STUDENT</ selector >
      < field >@S#</field>
</ key>
< keyref name="STUDENT_REFERENCE"
         refer= "STUDENT_KEY" >
      < selector >UNIVERSITY/DEPT/SUTENDT</ selector >
      < field >@STU_REF</ field >
</ keyref >
…
```