

Designing Semistructured Databases Using ORA-SS Model

Xiaoying Wu¹

Tok Wang Ling¹

Mong Li Lee¹

Gillian Dobbie²

¹School of Computing, National University of Singapore, Singapore
{wuxiaoy1, lingtw, leeml}@comp.nus.edu.sg

²Dept of Computer Science, University of Auckland, New Zealand
gill@cs.auckland.ac.nz

Abstract

Semistructured data has become prevalent with the growth of the Internet. The development of new web applications that require efficient design and maintenance of large amounts of data makes it increasingly important to design “good” semistructured databases to prevent data redundancy and updating anomalies. However, it is not easy, even impossible, for current semistructured data models to capture the semantics traditionally needed for designing databases. In this paper, we show how an Object-Relationship-Attribute model for SemStructured data (ORA-SS) can facilitate the design of “good” semistructured databases. This is accomplished via the normalization of ORA-SS. An XML DTD or Schema generated from a normal form ORA-SS schema diagram has no undesirable redundancy, and thus no updating anomalies for the complying semistructured databases. The general design methodology and detailed steps for converting an ORA-SS schema diagram into a normal form ORA-SS schema diagram are presented. These steps can also be used as guidelines for designing semistructured databases using the ORA-SS model.

1. Introduction

Semistructured data plays a crucial role in the new Internet applications ranging from electronic commerce to web site management to digital government. The emergence of XML (eXtended Markup Language) [3] as the likely standard for representing and exchanging data on the web has confirmed the central role of semistructured data. At the same time, XML has also redefined some of the ground rules. Perhaps the most important is that XML marks the “return of the schema”, in the form of Data Type Definition (DTD) and recently, XML-Schema [19], both of which are used to constrain valid XML documents. Many information providers have published their databases on the web as semistructured data, and others are developing repositories for new applications. This makes it important to have a guide for designing “good” semistructured databases. As with traditional databases, data redundancy and inconsistency

may occur in a semistructured database if its schema is not designed properly and thus will lead to undesirable anomalies.

It is a well known fact that data modeling is an inherent part of database design, dealing with the structure, organization and effective use of the information they represent [18]. However, current data models for semistructured data [1, 3, 4, 8, 14, 16] is inadequate in providing the semantics traditionally needed to fulfill the data modeling tasks. Although the Entity-Relationship data model is widely used in structured database design, it is not directly applicable to semistructured data.

This has motivated us to propose ORA-SS, an Object-Relationship-Attribute model for SemiStructured data [6]. ORA-SS is a semantically rich data model for semistructured data and comprises of four basic concepts: object classes, relationship types, attributes and references. It consists of four diagrams: the schema diagram, the instance diagram, the functional dependency diagram and the inheritance diagram. However, as traditional databases, ORA-SS schema diagrams may contain redundancies and suffer from undesirable updating anomalies. In relational databases, a series of database normal forms such as 3NF, 4NF and 5NF, has been proposed to determine whether a set of relations is a good design for a given database. For nested relations, normal forms like NNF (Nested Normal Form) [15] and NF-NR (Normal Form for Nested Relation) [11, 12] have been proposed to guarantee some good properties for the underlying databases. In [10], a normal form for Entity-Relationship diagram is proposed. One of the objectives of defining such normal form is to ensure that all the relations translated from ER diagram are in good normal form, either in 3NF or 5NF.

In this paper, we will define a normal form ORA-SS schema diagram. A normal form ORA-SS schema diagram ensures that the semistructured databases generated from the schema will have no undesirable redundancy and thus no updating anomalies. We will give a design methodology and present a comprehensive algorithm for normalizing an ORA-SS schema diagram into its normal form. The steps given in the algorithms can also be used as guidelines for designing semistructured databases using ORA-SS model.

The rest of the paper is organized as follows. Section 2 gives motivating examples. Section 3 briefly describes the ORA-SS model. An algorithm for mapping an ORA-SS schema diagram into XML DTD is also given. Section 3 defines the normal form ORA-SS schema diagram. Section 4 presents an algorithm for converting an ORA-SS schema diagram into a normal form. Section 5 discusses some related works and we conclude in Section 6 with directions for future work.

2. Motivation

Example 2.1 Consider the XML data in Figure 2.1(a). The details of a course are repeated for each professor that teaches the course. Figure 2.1(b) and (c) shows the corresponding ORA-SS instance and schema diagrams. There is a one-to-many binary relationship between *department* and *professor*, and a many-to-many binary relationship between *professor* and *course*. Note that the database instance in Figure 2.1(c) is not well designed because it contains redundancy: the same course information is repeated for each professor that teaches the course.

Similar to traditional databases, we can identify three kinds of update anomalies in a badly designed semistructured database: *insertion anomaly*, *rewriting anomaly* and *deletion anomaly* (see [17] for more details). The redundancy shown above can be avoided if *course* is referenced by a reference object class *course_i*, rather than nested within *professor*, as shown in Figure 2.2(a). When the semistructured database is based on this ORA-SS schema, the redundancy is eliminated (see Figure 2.2(b)).

In addition, there are more complex situations where the redundancy is harder, or even impossible, to recognize without knowing the semantics of data. Such situations occur in the presence of relationship attributes or n-ary relationship in semistructured data. Unfortunately, all the other data models proposed for XML, like DataGuide [8], ERX [16], ORM [1] and Xgrammar[14], stop short of dealing with these situations which are common in practice. The ORA-SS model is able to handle these, as we will illustrate in the following example.

Example 2.2 Consider the ORA-SS schema diagram in Figure 2.3(a). It contains a ternary relationship type *mp* between *project*, *member* and *publication*, and a binary relationship type *jm* between *project* and *member*. Figure 2.3(b) models an instance of this schema, showing the relationship between papers written by a particular member while working on a project will be nested within that member and project. From this diagram, we can deduce that publications *pub1* and *pub2* are associated with member *m1* and project *j1*. A DataGuide[8] for this schema is shown in Figure 2.3(c). However, if the relationship type *mp* is a binary relationship type between *member* and *publication*, which is represented by an ORA-SS schema diagram shown in Figure 2.3(d), then

there contains redundancy: all the publications for each member will be repeated for every project the member works on. Note that a DataGuide for the second schema will remain the same although the constraints on the relationship types are quite different. This distinction between binary and ternary relationship type cannot be expressed in other semistructured data models.

3. Background

In this section, we will give a brief description of ORA-SS schema diagram (see [6] for more details). We will also give an algorithm for mapping an ORA-SS schema diagram into XML DTD.

3.1 ORA-SS Model

Figure 3.1(a) shows an ORA-SS schema diagram. An object class is represented as a labeled rectangle. A relationship type between related object classes in an ORA-SS schema diagram can be described by *name*, *n*, *p*, *c*, where *name* (it is optional) denotes the name of the relationship type, *n* is an integer indicating the degree of the relationship type (*n*=2 indicates binary, *n*=3 indicates ternary, etc.), *p* is the participation constraint of the parent object class in the relationship type, and *c* is the participation constraint of the child object class. The participation constraints are defined using the *min*: *max* notation. Hence, 0:1, 0:n, 1:n represents ?,*,+ respectively. The edge between two object classes can have more than one such relationship type label to indicate the different relationship types they participate in. Disjunctive relationship type in ORA-SS is represented by a relationship type diamond labeled with symbol “|”.

Attributes of object class or relationship type are denoted by labeled circles. Keys are filled circles. An attribute can be single-valued or multivalued. A multivalued attribute is represented using an * or + inside the attribute circle. Attributes of an object class can be distinguished from attributes of a relationship type. The former has no label on its incoming edge while the latter has the name of the relationship type to which it belongs on its incoming edge. Note that an instance of that object class or relationship type would have a subset of the attributes shown.

An object class can reference another object class via a labeled and dashed edge. Such references are useful in modeling recursive and symmetric relationships.

3.2 Mapping ORA-SS Schema Diagram to XML DTD

Given an ORA-SS schema diagram, we can generate an XML DTD using the following algorithm.

Algorithm 1: Map ORA-SS Schema to XML DTD

Input: ORA-SS schema diagram *SD*;

Output: XML DTD

For each object class O in SD do:

Step 1. Generating definitions for object class

Generate element type definition

`<!ELEMENT O (subelementsList)>`.

Its sub-object classes (if any) become O 's sub-elements, whose names are contained in the O 's *subelementsList*. Sub-elements in *subelementsList* are separated by "|" if their corresponding object classes have disjunctive relationship with O as indicated by the diagram or by "," otherwise. No symbol is needed if there is only one sub-object class. We associate those sub-elements with frequency indicator such as ?, + or * , according to p of relationship type label *name*, n , p , c indicated by the diagram. In the case that O has no sub-object classes multivalued attributes and attached relationship attributes, then O 's *subelementsList* is #EMPTY.

Step 2. Generating definitions for attributes.

2.1 Generating definitions for single-valued simple attribute.

For each single-valued simple attribute a of O do

Generate attributes definition list `<!ATTLIST O attributeName type >` for O 's single-valued simple attributes.

The *type* for an attribute a is ID if a is O 's primary key; otherwise, its *type* is CDATA. If O is a reference object class¹, then define an attribute b with type IDREF and add it to O 's attributes definition list.

2.2 Generating definitions for single-valued composite attribute.

For each single-valued composite attribute a of O do

Replace a with its components and add those components to O 's attributes definition list

2.3 Generating definitions for multivalued simple attribute.

For each multivalued simple attribute a of O do

Generate an element type definition `<!ELEMENT a (#PCDATA)>`, and add the element name a to O 's *subelementsList*.

2.4 Generating definitions for multivalued composite attribute.

For each multivalued composite attribute a of O do

Generate an element type definition `<!ELEMENT a (#EMPTY)>`, and add the element name a to O 's *subelementsList*. For the components of a , generate attributes definition list `<!ATTLIST a componentName type >`.

Step 3. Generating definitions for relationship type attributes.

For each relationship type attribute A under O , add A to *subelementsList* in `<!ELEMENT O (subelementsList)>`.

Case (1) A is a simple attribute, generate an element type definition `<!ELEMENT A (#PCDATA)>`.

Case (2) A is a composite attribute, generate an element type definition `<!ELEMENT A (#EMPTY)>`. For the components of A , generate attributes definition list `<!ATTLIST A componentName type >`.

Example 3.1 An XML DTD for the ORA-SS schema diagram in Figure 3.1(a) is shown in Figure 3.1(b).

3.3 XML's Inadequacies

The popularity of using XML to model semistructured, hierarchical data on the web encourages the view of XML as a data model [4]. However, the mapping process given in Algorithm 1 reveals that, from the database aspects of view, XML has very restrictive definitions and has several drawbacks. First, although attributes of IDREFS type can be viewed as multivalued attributes, other kinds of multivalued attributes are not allowed in XML's structure. They have to be converted to sub-elements. Hence, when we translate an ORA-SS schema diagram to XML DTD, the semantics of the real world is lost and ambiguity is generated. Second, the concept of composite attributes is not included in XML. They either have to be replaced by their components or be converted to sub-elements. Hence, XML has imprecise definitions and cannot handle the consequent ambiguities as well. Note that these are inherent shortcomings of XML that limit the data description capabilities of its schema definition languages, including DTD or XML Schema. So we argue that using XML is awkward to represent all the necessary semantics for modeling real world data, unless it can incorporate the concepts of multivalued attribute and composite attribute to its structure. In contrast, by allowing the existence of multivalued attributes and composite attributes, ORA-SS removes the aforementioned drawbacks. Additionally, the ability of ORA-SS to express the degree of n -ary relationship types, and distinguish between attributes of object classes and attributes of relationship types helps us to recognize redundancy, design more efficient storage and access to data and define meaningful views [11].

4. Normal Form (NF) for ORA-SS Schema Diagram

ORA-SS is similar to nested relations in that both have tree-like structure and allow repeating groups or multiple occurrences of objects. Hence, starting from the top of a given ORA-SS schema diagram D , we can easily construct a nested relation R , which has the single valued attributes of D 's root object class as its atomic attributes, and the multivalued attributes as well as the sub-object classes of D 's root object class as its repeating groups. As an illustration, the corresponding nested relation for the ORA-SS schema diagram in Figure 2.1 is *Department (d-name, course (code, title, student (number, s-name, grade)*))**. We can construct a set of nested relations for an ORA-SS schema diagram that consists of several separated tree-structured components (each starts from different roots and perhaps related to others through reference semantics).

OEM[8] is a popular data model for representing semistructured data. We have seen an example of its

¹ A *reference object class* in an ORA-SS schema diagram is an object class which has no properties of its own and has to references properties of another object class. The reference semantics is represented as a dashed edge between the two object classes in the schema diagram.

schema DataGuide. OEM is a simple model, with every entry as object identified by a 3-tuple: <object-identifier (OID), label, value>. Like ORA-SS, a database is represented as a tree-structured graph.

While the three data models can represent hierarchical data in a direct and natural way, they have problems when representing situations with nonhierarchical relationships. Duplication of data is necessary when we try to represent many-to-many relationships or relationships involving more than two participating entity types or object classes. The situation may be even worse for OEM in the following reasons: First, except the nested structure of the data, other semantic information cannot be modeled using a Dataguide; second, the use of OID in OEM incurs problems as those in object-oriented model. As in traditional databases, redundancy leads to possible update anomalies in semistructured data. Until a normalization theory is defined for semistructured data, the best way to identify and eliminate redundancy is to use heuristics.

The correspondences between ORA-SS schema diagrams and nested relations suggest that we can define an ORA-SS schema diagram in normal form if its corresponding set of nested relations is in *normal form for set of nested relations*, which has been defined in [11, 12]. This in turn ensures that semistructured databases, which conform to an XML DTD generated from a normal form ORA-SS schema diagram, can have no redundancy and no undesirable updating anomalies.

The concept of a normal form (NF) ORA-SS schema diagram depends on the twin concepts of an object class normal form (O-NF) and a relationship type normal form (R-NF). [10] defines entity and relationship normal forms for an Entity-Relationship diagram. The results there can be applied here with some modification to account for ORA-SS's tree-like structure.

Definition 4.1 An object class O of an ORA-SS schema diagram is said to be in *object class normal form* (O-NF), if the nested relation constructed by O 's single valued attributes as its atomic attributes, O 's multivalued attributes as its repeating groups, is in normal form NF-NR.

Definition 4.2 A relationship type R of an ORA-SS schema diagram D is said to be in *relationship type normal form* (R-NF), if the nested relation constructed by the keys of the participating object classes, and R 's atomic attributes as its atomic attributes, R 's multivalued attributes as its repeating groups, is in normal form NF-NR.

The reasons for the conditions of Definition 3.1 as well as Definition 3.2 have been well explained in [10, 11, 12].

Definition 4.3 An ORA-SS schema diagram D is in *normal form* (NF) iff it satisfies the following conditions:

1. Every object class in D is in O-NF.

2. For every relationship type R in D

- (a) R is in R-NF.

- (b) *Case (1)* If R is binary relationship type from object class A to object class B , then all the B 's attributes can stay with B only if R is a one-to-many or one-to-one binary relationship type from A to B . All the attributes of R (if any) should be attached to B .

- Case (2)* If R is n -ary relationship type with n ($n > 2$) participating object classes O_1, O_2, \dots, O_n , and the path going downward from the top of D linking those object classes is $/O_1/O_2/\dots/O_n$, then for each object class O_i ($2 \leq i \leq n$),

- (i) O_i should have an i -ary relationship R_i with its ancestors O_1, O_2, \dots, O_{i-1} .

- (ii) The attributes of O_i can stay with O_i only if functional dependency $O_i \rightarrow O_1, O_2, \dots, O_{i-1}$ can be derived from the functional dependency diagram for D . The attributes of R_i (if any) should be attached to O_i .

3. There is no relationship type nested under another many-to-many or many-to-one binary or n -ary ($n > 2$) relationship type.

4. Every relationship type cannot be derived from other relationship types in D .

In Definition 4.3, Case (1) and item (ii) of Case (2) in condition 2(b) ensure that there will be no potential redundancies due to many-to-many and n -ary relationship representation; Item (i) of Case (2) in Condition 2(b) helps to remove *over-nesting*. Intuitively, components should be kept as close to the *owner* object class as possible. Condition 3 helps to reduce data redundancy as well as ensure no unnecessary hierarchies in a schema diagram. Condition 4 removes global redundancies among a set of components in a NF schema diagram. Note that other normal forms proposed for semistructured data, like S3-NF [9] and XNF [7] do not provide similar definitions.

If an ORA-SS schema diagram is in normal form, then the anomalies in semistructured databases mentioned in Example 2.1 are removed and any redundancy due to many-to-many relationships and n -ary relationships are controlled.

Example 4.1 Consider the *staff* object class given in Figure 4.1(a). Assume we have following functional dependencies: $\{S\# \rightarrow dept, dept \rightarrow faculty\}$, then obviously, the relation *staff* ($S\#, dept, faculty$) is not in 3NF, so is not in NF-NR [11,12]. Hence the condition of O-NF definition is violated, and *staff* is not in O-NF.

Example 4.2 Consider the ORA-SS schema diagram given in Figure 4.1(b). The schema attempts to show that the lecturer can teach all the courses using all the textbooks as described on the curriculum, and is designed as a ternary relationship among *course*, *text* and *lecture*. However, it is a wrong design, since by the condition, a course taught by a teacher is independent of the textbook

used, i.e., a MVD constraints: $course-code \twoheadrightarrow isbn \mid staff\#$ should be satisfied by the schema. Hence, while the nested relation ctl ($course-code, isbn, staff\#$) for the relationship type ctl is in 3NF, it is not in 4NF, so as not in NF-NR; the condition of R-NF definition is violated, and ctl is not in R-NF.

Example 4.3 Consider the ORA-SS schema diagram given in Figure 4.2(a). If examined individually, the schema diagrams for both *faculty* and *employee* are all in NF. However, suppose that a *faculty* is also an *employee*, the schema for the database is not in normal form since the qualification of *faculty* can be derived from that of *employee*. As a consequence, *qualification* information for a *faculty* will be repeated in the underlying databases. A better design is to remove the *qualification* from *faculty*, and make *ssn* of *faculty* as a foreign key that references *employee*, as shown in Figure 4.2(b).

5. Converting ORA-SS Schema Diagrams into Normal Form

There are two approaches for designing semistructured databases. The first approach is based on the users' requirements, first we come out an initial ORA-SS schema diagram; After that, we normalize the schema diagram to its normal form; Finally, we map the normalized schema to an XML DTD using Algorithm 1. The second approach is, given a semistructured data instance, like an XML document, we can design it using the following steps:

- (1) Extract schema from the instances using the schema extracting techniques, like what is given in [2];
- (2) Translate the schema into ORA-SS schema diagram. Here we need semantic enrichment, since not all semantics needed are available from the extracted schema.
- (3) We convert the ORA-SS schema diagram into its normal form.
- (4) We translate the NF ORA-SS schema diagram back to XML DTD or XML Schema.
- (5) Restructuring the initial instance to conform to the generated XML DTD or XML Schema.

In this paper, we focus on the first design approach.

The following conversion algorithm takes as input an ORA-SS schema diagram and functional dependency diagram², and returns as output an NF ORA-SS schema diagram. The design steps are given to achieve the definitions of NF ORA-SS schema diagram

Algorithm 2: Converting an ORA-SS schema diagram into NF ORA-SS schema diagram.

Input: an ORA-SS schema diagram SD , and its functional dependency diagram.

Output: a NF ORA-SS schema diagram.

² In the interest of space, we don't provide functional dependency diagram in this paper.

Step 1. For each non O-NF object class O in SD , convert O into O-NF, using the guidelines and steps given in [10].

Step 2. Make each relationship type R in R-NF, using the guidelines and steps given in [10].

Step 3. This step involves two sub-steps.

3.1 Construct diagrams for each object class with its attributes in SD .

3.2 For each relationship type R in SD do

Case 1: R is a binary relationship type from object class O_A to O_B . Assume R is described by a relationship type label L with contents $name, 2, p, c$ in SD . Basing on O_A ³, we represent R as follows:

If R is an one-to-many ($O_B \rightarrow O_A$) or one-to-one relationship type, then

(a) Nest O_B along with its attributes under O_A , and tag the edge between them with L ;

(b) Attach all the attributes of R to O_B , and tag the edges between attributes and O_B with the name R ; $O_B \rightarrow O_A$

Else /* R is many-to-one ($O_A \rightarrow O_B$) or many- to-many relationship type*/

(a) Construct a reference object class O'_B ⁴ referencing O_B , and nest O'_B under O_A . Tag the edge between O_A and O'_B with L .

(b) Attach all the attributes of R to O'_B , and tag the edges between attributes and O'_B with the name R ;

Case 2: R is an n -ary relationship type where $n > 2$ with participating object classes O_1, O_2, \dots, O_n . Let the path that links those object classes by going down the SD be $/O_1/O_2/\dots/O_n$. Let R_i ($2 \leq i \leq n$) represents the relationship from R_{i-1} to O_i , (if $i = 2$, then R_{i-1} is O_1), then R can be represented by a sequence of relationships $\langle R_1, R_2, \dots, R_n \rangle$. Assume each R_i is described by a relationship type label L_i with contents $name, i, p, c$ in SD . We represent each R_i based on R_{i-1} ($3 \leq i \leq n$)⁵ as follows:

If $O_i \rightarrow O_1, O_2, \dots, O_{i-1}$ can be derived from the specified dependency constraints for SD , then

(a) Nest O_i along with its attributes under R_{i-1} , and tag the edge between them with L_i ;

(b) Attach all the attributes of R_i to O_i , and tag the edges between attributes and O_i with the name R_i ;

Else /* $O_i \not\rightarrow O_1, O_2, \dots, O_{i-1}$ */

(a) Construct a reference object class O'_i referencing O_i , and nest O'_i under R_{i-1} . Tag the edge between R_{i-1} and O'_i with L_i .

(b) Attach all the attributes of R_i to O'_i , and tag the edges between attributes and O'_i with the name R_i ;

Step 4. If a relationship type R is redundant, then the information provided by R can be derived from other relationship type, such that data will be redundant in the underlying databases. To detect the redundant relationship type, we require more information about the semantic

³ In the algorithm, we let the object class name denote its corresponding diagram constructed in Step 3(a).

⁴ It is generally preferable to have designers/users specify alternate names for referencing object class, which indicate the role played by the object class in the context of the application. Here for simplicity, we assume the name of referencing object class is that of referenced object class append with subscript number like 1, 2 etc.

⁵ We start i with 3, since the relationship type R_2 has been represented by case 1.

meaning of the relationship types, which can be provided by the database designer or database owner.

Theorem 1 Let S be an ORA-SS schema diagram generated by Algorithm 2, then it is a normal form ORA-SS schema diagram.

Proof: Omitted. Details can be found in [20].

Example 5.1 Consider the ORA-SS schema diagram D represented in Figure 5.1(a). There is a many-to-many binary relationship pc between *professor* and *course*, and a many-to-many binary relationship ct between *course* and *textbook*. Applying Algorithm 2 to this ORA-SS schema diagram, we observe that the three object classes *course*, *student* and *tutor* are all in O-NF; the two binary relationship type cs and cst are both in R-NF; hence step 1 and 2 are passed. Starting from step 3, we first generate three diagrams for the object classes with attributes as shown in Figure 5.1(b). Next, we represent the binary relationship pc . Since pc is a many-to-many relationship type from *professor* to *course*, we create a reference object class $course_1$ referencing *course* and nest $course_1$ under *professor*, as shown in Figure 5.1(c). After that, we represent the binary relationship ct . Since ct is a many-to-many relationship type from *course* to *textbook*, we create a reference object class $textbook_1$ referencing *textbook* and nest $textbook_1$ under *course*, as shown in Figure 5.1(d). Since there is no redundant relationship type, the schema diagram in Figure 5.1(d) is in normal form.

Example 5.2 Consider the ORA-SS schema diagram D in Figure 5.2(a), assume the specified functional dependency is $\{student, course \rightarrow tutor\}$. There is a binary relationship cs between *course* and *student* and a ternary relationship cst between *course*, *student* and *tutor*. The *grade* is an attribute of the binary relationship cs , and *feedback* is an attribute of the ternary relationship cst . Applying Algorithm 2 to D , we observe that the three object classes *professor*, *course* and *textbook* are all in O-NF; the two binary relationship type pc and ct are both in R-NF; hence step 1 and 2 are passed. Starting from step 3, we first get three diagrams for object classes *course*, *student* and *tutor*, as shown in Figure 5.2(b). Next, we represent the binary relationship cs . Since cs is a many-to-many relationship type from *course* to *student*, we create a reference object class $student_1$ referencing *student* and nest $student_1$ under *course*. Relationship attribute *grade* is attached to $student_1$. The result is shown in Figure 5.2(c). After that, based on the relationship cs , we represent the relationship cst according to case 2 of step 3. Since $tutor \rightarrow student$, *course* cannot be derived from the given functional dependency, we create a reference object class $tutor_1$ referencing *tutor*, and nest $tutor_1$ under $student_1$. Relationship attribute *feedback* is attached to $tutor_1$, as shown in Figure 5.2(d). Since there is no redundant relationship type, the diagram shown in Figure 5.2(d) is now in normal form.

6. Related Work

To our knowledge, two normal forms for semistructured data have been proposed: S3-NF in [9], and most recently XNF in [7].

S3-NF is a normal form for S3-Graph (or SemiStructured Schema Graph), which is basically a labeled graph in which vertices correspond to objects and edges represent the object-subobject relationship. Unlike ORA-SS schema diagram, the S3-Graph is not able to model the semantics traditionally needed for recognizing redundancy in databases. For example, it cannot show the degree of a n -ary relationship type, neither can it distinguish between attributes of object classes and attributes of relationships types. To identify redundancy in S3-Graph, [9] defines a dependency constraint called SS-Dependency. An S3-Graph is in S3-NF if there is no transitive SS-dependency. Hence, only that kind of redundancy can be recognized by S3-NF. [9] presents two approaches to design S3-NF databases. One is a decomposition method, which can transform the schema to reduce redundancy result from SS-dependency, while may not always remove all transitive dependencies and achieve normal form. The other method is to transform a normal form ER diagram [10] into an S3-Graph. Although the result obtained is in S3-NF, it is not unique but is dependent on the path constructed. Therefore, the result may not satisfy the application requirements and comply with the user's viewpoints.

XNF is defined to be a normal form for XML documents [7]. The whole process of generating an XNF-compliant DTD follows: it first takes a conceptual model - based methodology, using CM hypergraphs (conceptual-model hypergraphs), to model an application. Then it translates the CM hypergraph M to a scheme-tree forest F . F is in XNF if each scheme tree in F has no potential redundancy with respect to a specified set of (functional and multivalued) constraints C , and F has as few, or fewer, scheme trees as any other schemes-tree forest corresponding to M in which each scheme tree has no potential redundancy with respect to C . Finally, it generates a DTD from the scheme-tree. Like S3-Graph, CM hypergraph has no concept of attributes; consequently, there are too many objects in a schema; in addition, CM hypergraph has no hierarchical structure. The algorithms for translating a CM hypergraph M to a scheme-tree forest are non-deterministic, and suffer from inefficiency. Additionally, adding new required information requires redesign the whole schema. Further, the algorithms generate a large number of solutions rather than verifying whether a semistructured schema is in normal form or not. While ISA relationship can be represented in CM hypergraphs, it is from CM hypergraph before input to the algorithm.

The normal form ORA-SS schema diagram presented in this paper has two advantages over both S3-NF and XNF. Firstly, ORA-SS facilitates the 2-level design

technique: First, designer identifies or figures out object classes and relationship types from user's specifications; then the designer add attributes for object classes and relationship types. The 2-level design technique is consistent with the iterative nature of ER designing methodology, giving more control to the designer and allows him/her to evaluate each successive refinement of the schema. Secondly, ORA-SS designing approach is able to preserve a schema's hierarchical structure satisfying the user's requirements.

5. Conclusion

In this paper, we have demonstrated that the ORA-SS data model beats all the existing semistructured data models for its ability to design databases, and thus makes itself an attractive candidate for logical semistructured database design. We have identified various anomalies, including rewriting anomaly, insertion anomaly and deletion anomaly, which may arise if a semistructured database is not designed properly and contains redundancies. We have defined a normal form ORA-SS schema diagram. The definitions of a normal form ORA-SS schema diagram give the necessary and sufficient conditions for ensuring the corresponding set of nested relations in normal form for set of nested relations [11,12]. This in turn ensures that semistructured databases conforming to an XML DTD, which is generated from a normal form ORA-SS schema diagram, can have no unnecessary redundancy and thus no undesirable updating anomalies. We have presented a general designing methodology and developed an algorithm for converting a given ORA-SS schema diagram into its norm form. The steps presented can also be used as guidelines for designing semistructured databases using the ORA-SS model. For future work, we would like to implement a case tool based on the ORA-SS model for designing semistructured databases.

Reference

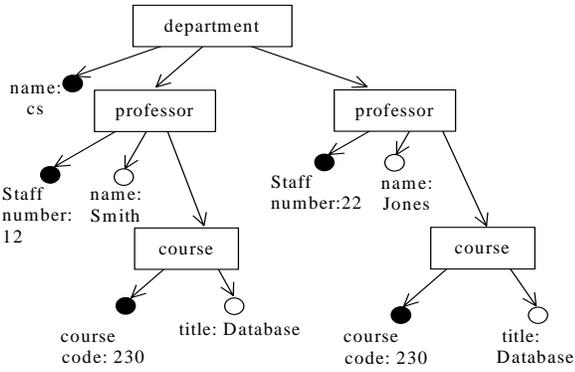
- [1] L. Bird, A. Goodchild, and T.Halpin. Object Role Modeling and XML-Schema. In *Int'l Conf. on Conceptual Modeling (ER)*, Salt Lake City, UT, Oct.2000
- [2] B. Ludaescher, Y. Papakonstantinou, P. Velikhov, and V.Vianu. View definition and dtd inference for xml. In *Workshop on semistructured Data and Nonstandard Data Formats*, January 1999.
- [3] T. Bray, J. Paoli, and C. M. Sperberg-McQueen. Extensible Markup Language (XML) 1.0. 2nd Edition, Oct. 2000. <http://www.w3.org/TR/REC-xml>.
- [4] P. Buneman, S. B. Davidson, M. F. Fernandez and D. Suciu: Adding Structure to Unstructured Data. *ICDT 1997*: 336-350.
- [5] S. Ceri, P. Fraternal, and S. Paraboschi. XML: Current Developments and Future Challenges for the Database Community. *EDBT 2000, LNCS 1777*, pp.3-17, 2000.
- [6] G.Dobbie, X.Y.Wu, T.W.Ling and M.L.Lee. ORA-SS: An Object-Relationship-Attribute Model for Semistructured Data. Technical Report TR21/00, School of Computing, National University of Singapore, 2000.
- [7] D.W.Embley and W.Y.Mok. Developing XML Documents with Guaranteed "Good" Properties. *ER 2001*.
- [8] R. Goldman and J. Widom. DataGuides: Enabling Query Formulation and Optimization in Semistructured Databases. *Proceedings of the Twenty-Third International Conference on Very Large Data Bases*, pages 436-445, Athens, Greece, August 1997.
- [9] S. Y. Lee, M. L. Lee, T. W. Ling and L. A.. Kalinichenko. Designing Good Semi-structured Databases. *ER 1999*: 131-145
- [10] T.W. Ling. A Normal Form for Entity-Relationship Diagrams. *Proc. 4th International Conference on Entity-Relationship Approach* (1985)
- [11] T. W. Ling. A normal form for sets of not-necessarily normalized relations. In *Proceedings of the 22nd Hawaii International Conference on System Sciences*, pp. 578-586. United States: IEEE Computer Society Press, 1989.
- [12] T. W. Ling and L. L. Yan. NF-NR: A Practical Normal Form for Nested Relations. *Journal of Systems Integration*. Vol4, 1994, pp309-340
- [13] T. W. Ling, M. L. Lee and G.Dobbie. Applications of ORA-SS: An Object-Relationship-Attribute data model for Semistructured data. In *Third International Conference on Information Integration and Web-based Applications and Services (IIWAS2001)*.
- [14] Murali Mani, Dongwon Lee and Richard R. Muntz Semantic Data Modeling using XML Schemas *Proc. 20th Int'l Conf. on Conceptual Modeling (ER)*, Yokohama, Japan, November, 2001.
- [15] Z. M. Ozsoyoglu and L. Y. Yuan. A New Normal Form for Nested Relations. *ACM Transaction on Database Systems*. 12(1), (1987).
- [16] G. Psaila. ERX: A data model for collections of XML documents. In *ACM Symp. On Applied Computing (SAC)*, Villa Olmo, Italy, Mar. 2000
- [17] R. Ramakrishnan and J.Gehrke. *Database Management Systems*. McGraw-Hill Higher Education, 2000.
- [18] D. T. Chritzis and F. H. Lochovsky. *Data Models*. Prentice-hall, 1982
- [19] H. S. Thompson, D. Beech, M. Maloney and N. Mendelsohn. XML Schema Part 1: Structures. Oct. 2000 <http://www.w3.org/TR/xmlschema-1/>.
- [20] X. Wu. The Design of Semistructured Schemata into Good Normal Form. Master Thesis, School of Computing, National University of Singapore, 2001.

```

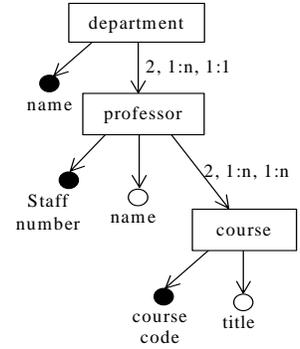
<department>
  <name>cs</name>
  <professor>
    <staffnumber>12</staffnumber>
    <name>Smith</name>
    <course>
      <coursecode>230</coursecode>
      <title>Database</title>
    </course>
  </professor>
  <professor>
    <staffnumber>22</staffnumber>
    <name>Jones</name>
    <course>
      <coursecode>230</coursecode>
      <title> Database</title>
    </course>
  </professor>
</department>

```

(a) Example of XML data

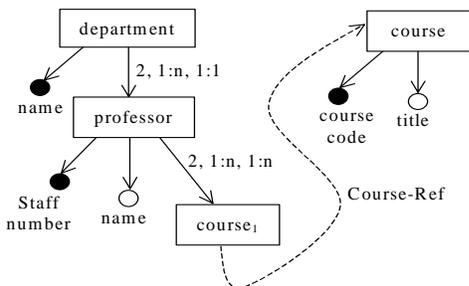


(b) ORA-SS instance diagram

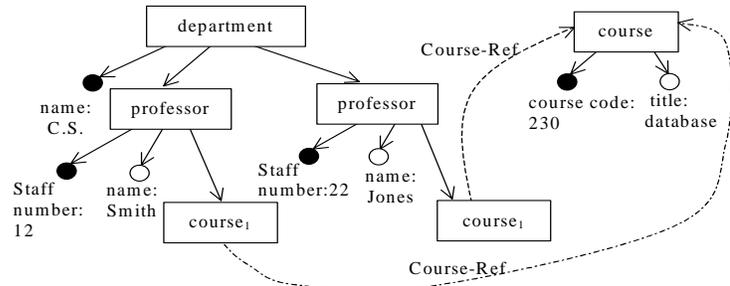


(c) Nested object class in an ORA-SS schema diagram

Figure 2.1: Redundancy in Semistructured data

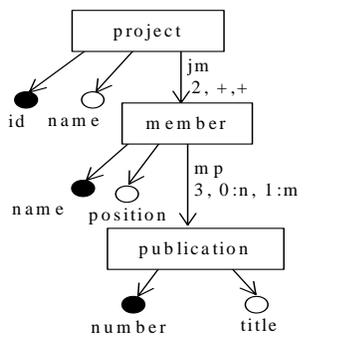


(a) Referenced object class in ORA-SS schema diagram

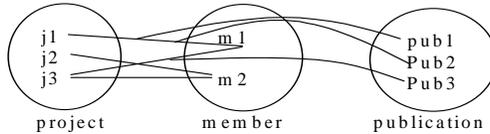


(b) The ORA-SS instance diagram

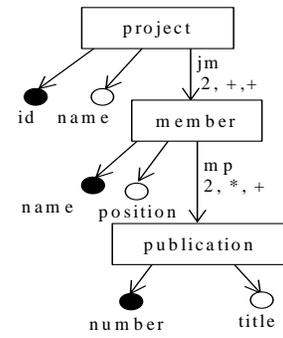
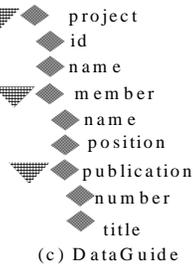
Figure 2.2: Referenced object classes in ORA-SS schema diagram and instance diagram



(a) ORA-SS Schema Diagram (*mp* is a ternary relationship type)

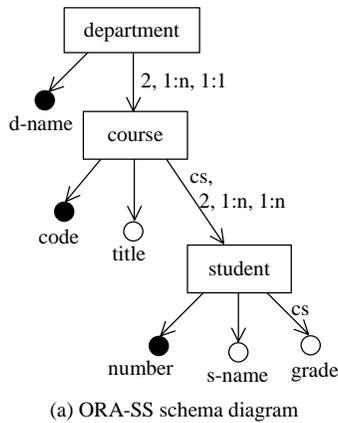


(b) A data instance of (a)



(d) ORA-SS Schema Diagram (*mp* is a binary relationship type)

Figure 2.3: Representing ternary relationship in an ORA-SS Schema Diagram



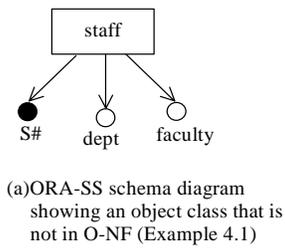
```

<!ELEMENT department (course+)>
<!ATTLIST department d-name ID #REQUIRED>
<!ELEMENT course (student+)>
<!ATTLIST course code ID #REQUIRED
title CDATA>
<!ELEMENT student (grade)>
<!ATTLIST student number ID #REQUIRED
s-name CDATA>
<!ELEMENT grade (#PCDATA)>

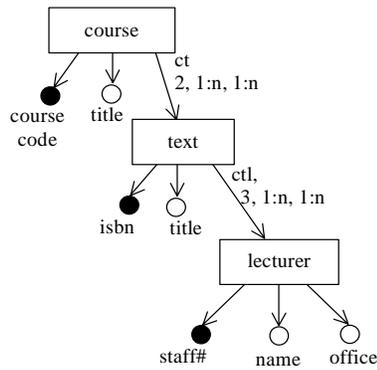
```

(b) The corresponding XML DTD

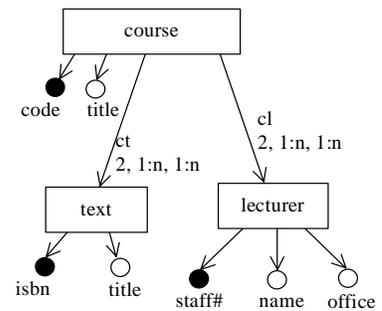
Figure 3.1: ORA-SS schema diagram and its XML DTD specification



(a) ORA-SS schema diagram showing an object class that is not in O-NF (Example 4.1)

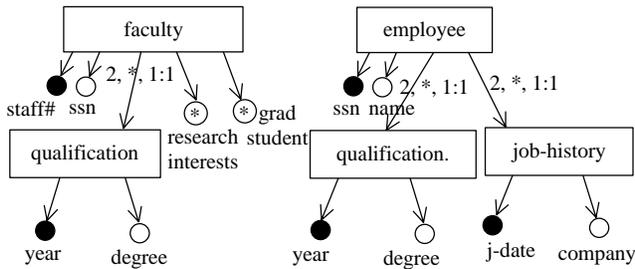


(b) ORA-SS schema diagram not in NF (Example 4.2)

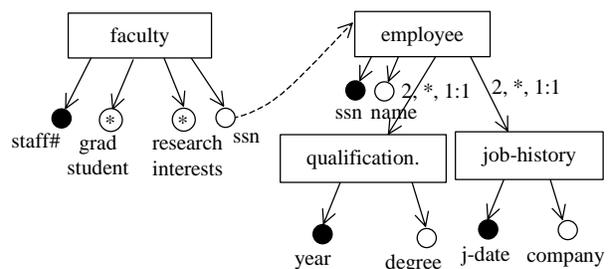


(c) A better design for (b): No "over-nesting" No unnecessary redundancy

Figure 4.1: ORA-SS schema diagrams



(a) Un-normalized ORA-SS schema diagram: Since a *faculty* is also an *employee*, thus the *qualification* of *faculty* can be derived from that of *employee*



(b) Normalized ORA-SS schema diagram for (a)

Figure 4.2: ORA-SS schema diagrams for example 4.3

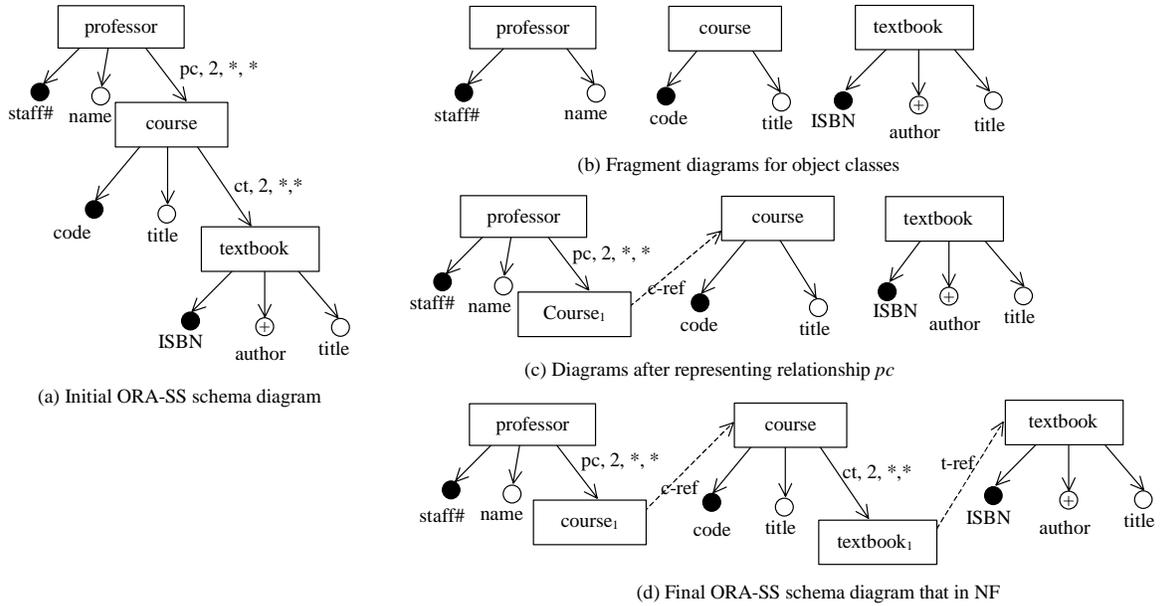


Figure 5.1: Figures for the example 5.1 illustrating Algorithm 2

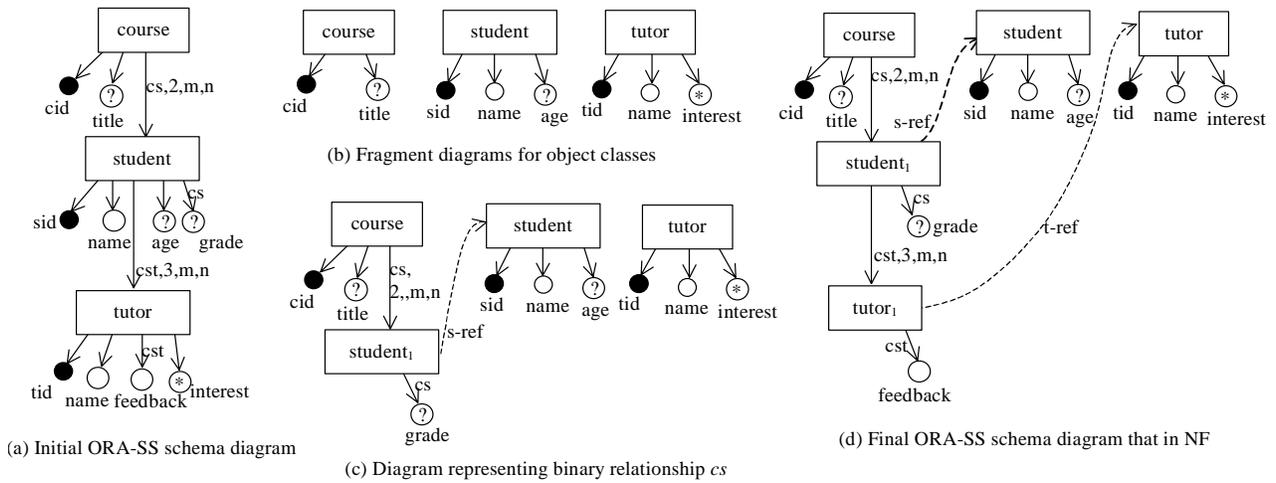


Figure 5.2: Figures for the example 5.2 illustrating Algorithm 2