

Maintaining Semantics in the Design of Valid and Reversible SemiStructured Views

Ya Bing Chen

Tok Wang Ling

Mong Li Lee

School of Computing, National University of Singapore
(chenyabi, lingtw, leeml)@comp.nus.edu.sg

Abstract. Existing systems that support semistructured views do not maintain semantics during the process of designing views. Thus, there is no guarantee that the views obtained are valid and reversible views. In this paper, we propose an approach to designing valid and reversible semistructured views. We employ four types of view operators, namely, *select*, *drop*, *join* and *swap* operators, and develop a set of rules to maintain the semantics of the views when the *swap* operator is applied. We also examine the reversible view problem and develop rules to guarantee the designed views are reversible. Finally, we examine the possible changes to the participation constraints of relationship types and propose rules to keep the participation constraints correct.

1 Introduction

Existing systems [1, 2, 3, 4, 7, 9, 10, 11, 12] for semistructured views do not maintain semantics during the process of designing views. Thus, they cannot guarantee the validity and reversibility of the views. [5, 6] propose a novel approach to design valid views over semistructured data. A conceptual schema for source data is first extracted based on a semantically rich data model, the Object-Relationship-Attribute model for Semi-Structured data (ORA-SS) [8]. Semistructured views are created by applying four transformation operators on the source ORA-SS schema. The operations are *select*, *drop*, *join* and *swap*.

In this paper, we develop a complete set of rules to ensure that the designed views are meaningful and reversible back to the original source schema when the *swap* operator is applied. We also develop additional rules to keep the participation constraints correct for relationship types in the views. To the best of our knowledge, this is the first work to address the problem of maintaining semantics in the design of valid and reversible semistructured views.

The rest of the paper is organized as follows. Section 2 reviews the ORA-SS data model and highlight the semantics that are captured in ORA-SS for the design of valid and reversible semistructured views. Section 3 presents the design rules for the *swap* operator. Additional rules for the evolution of the participation constraints of object classes in relationships are given in Section 4, and we conclude in Section 5.

2 Motivating Example

The ORA-SS model comprises of three basic concepts: *object classes*, *relationship types* and *attributes*, and captures richer semantics compared to models such as OEM and XML DTD/Schema.

Example 1. Figure 1 depicts an ORA-SS source schema. This schema has 4 object classes: *course*, *student*, *lecturer* and *tutor*. Each object class has a key attribute, which is denoted by filled circle. The attribute *hobby* of *student* is a multi-valued attribute which is indicated by an asterisk inside the circle. There is a binary relationship type between object class *course* and *student*, which is labeled as “*cs*, 2, 1:n, 1:n” on the incoming edge of *student*. In the label, *cs* denotes the name of the relationship type, 2 indicates the degree of the relationship type, the first “1:n” indicates the parent participation constraints in the relationship type, and the second “1:n” indicates the child participation constraints in the relationship type. The relationship type has an attribute *grade* attached to *student* with label *cs* on the incoming edge of the attribute, which implies a functional dependency $stuNo, code \rightarrow grade$. Similarly, there is a ternary relationship type “*cst*, 3, 1:n, 1:n” labeled on the incoming edge of *tutor*. The relationship type involves three object classes *course*, *student* and *lecturer*. In this case, the first “1:n” indicates the participation constraints of the relationship type *cs* in the ternary relationship type. The second “1:n” indicates the participation constraints of the object class *tutor* in the ternary relationship type. □

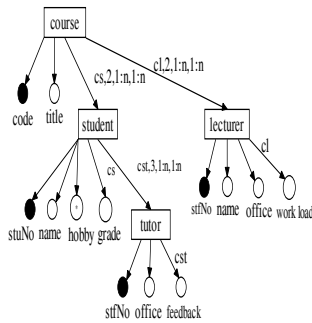


Fig. 1. ORA-SS source schema

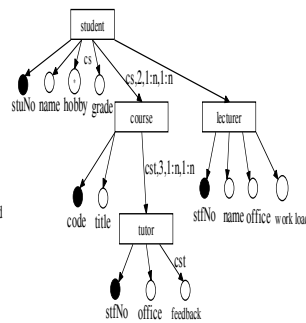


Fig. 2. Invalid ORA-SS view obtained by swapping *course* and *student* in Figure 1

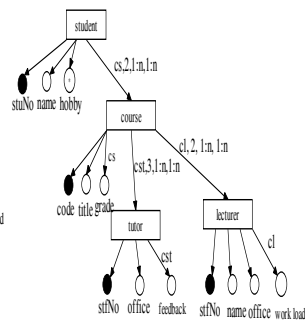


Fig. 3. Valid reversible ORA-SS view obtained by swapping *course* and *student* in Figure 1

Based on the ORA-SS schema in Figure 1, let us design a view by swapping *course* and *student*. The attributes of the two object classes will move with their owner object classes respectively. The attribute *grade* which belongs to the relationship type *cs* will move with *student* if we design the view based on XML DTD/Schema or OEM graph. This is because XML DTD/Schema or OEM do not differentiate between attributes of object classes and attributes of relationship types. For illustration purposes, the view in Figure 2 attaches *grade* to *student* to show such a case, which violates the functional dependency implied in the source schema, i.e., $stuNo, code \rightarrow grade$. The view in Figure 2 is invalid since it violates the semantics in the source schema.

The above example shows that invalid views arise when important semantics are not expressed in the underlying data model. Figure 3 shows a valid ORA-SS view when the *swap* operator is applied on the same source schema in Figure 1. In Figure 3, the attribute *grade* is moved down and attached to *course* to keep the functional dependency of the relationship type *cs* intact. The object class *lecturer* also needs to move down with *course* to keep all the three participating object class of *cst* in one hierarchical path. Thus, the semantics of the relationship type *cst* is kept intact.

3 Design Rules for Swap Operator

Valid semistructured views considered in this work can be obtained by applying four view operators on an ORA-SS schema. The four operators are *select*, *join*, *drop* and *swap*. As a subset of XQuery, these operators fulfill most of the data-centered query requirements for XML. [5] develops a set of rules to guarantee the validity of semistructured views when the operators *select*, *join*, *drop* are applied, and gives an initial proposal of how the validity of views can be maintained when *swap* operator is applied. In this section, we will detail how valid semistructured views can be designed with *swap* operator. The *swap* operator restructures an ORA-SS source schema by exchanging the positions of a parent object class and its child object class.

Rule Swap_1: *If an object classes O_i and its descendant object class O_j in a source schema are swapped in designing a view; then the attributes of O_i and O_j must remain attached to O_i and O_j respectively in the view.*

Rule Swap_1 is straightforward and ensures that the attributes of two object classes O_i and O_j do not become meaningless in the view after O_i and O_j are swapped. More importantly, we observe that the relationship types in an ORA-SS source schema that involve O_i and/or O_j are also affected since the hierarchical positions of O_i and O_j have been interchanged after a *swap* operator is applied. Given two object classes O_i and O_j where O_j is a descendant of O_i in an ORA-SS schema, the relationship types that are affected after a swap of O_i and O_j can be classified into three categories.

The first category is the set of relationship types which do not involve any other object classes but O_i and/or O_j and/or the ancestors of O_i or O_j in the ORA-SS source schema. In another words, these relationship types involve object classes that occur in the straight path of O_i and O_j (see Figure 4).

The second category is the set of relationship types which involve at least both O_i and object classes in the branch paths between O_i and the parent of O_j , as shown in Figure 5. The third category is the set of relationship types which involve at least both O_j and its descendants, as shown in Figure 6. The three categories of affected relationship types are handled by the rules Swap_2, Swap_3 and Swap_4 respectively.

Rule Swap_2: *Suppose an object classes O_i and its descendant object class O_j in a source schema S are swapped in designing a view. Let S be the set of relationship types which do not involve any other object classes but O_i and/or O_j and/or the ancestors of O_i or O_j in the source schema. For each relationship type R in S , the attributes of R are attached to the lowest participating object class of R in the view.*

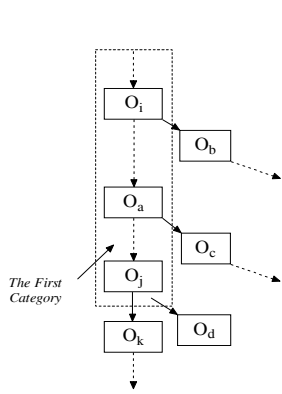


Fig. 4. First category of relationship types.

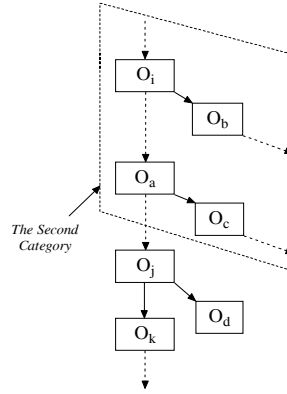


Fig. 5. Second category of relationship types.

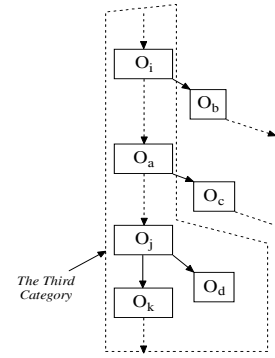


Fig. 6. Third category of relationship types.

Rule Swap_3: Suppose an object class O_i in a source schema is swapped with its descendant object class O_j in designing a view. If there exists a relationship type which involves at least O_i and O_c , where O_c is a descendant of an object class O_a that lies in the path between O_i and O_j (including O_i) but O_c does not lie in the path between O_i and O_j in the ORA-SS source schema, then the subtree rooted at O_c is attached to O_i in the view.

Rule Swap_4: Suppose an object class O_i in a source schema is swapped with its descendant object class O_j in designing a view. For each child O_a of the object class O_j , let T be the subtree that is rooted at O_a . Let S be the set of relationship types which involve at least O_j and its descendants in T . If O_l is the lowest participating object class among all the relationship types in S that lie in the path between O_i and O_j after the swap, then the subtree rooted at O_a is attached to O_l in the view.

The swap operator also introduces the issue of reversible views in semistructured data. A valid view schema V of a source schema S is called a reversible view if S can be produced back from V through applying our view operators, i.e. *select*, *drop*, *join* and *swap*. Here, we only consider *swap* operator for the issue of reversible view.

We observe that the rules Swap_3 and Swap_4 not only maintain the semantics of a semistructured view so that it is kept valid, but they also guarantee that the view is reversible. For example, let us now apply another *swap* operator to the view in Figure 3 to swap *student* and *course* again. Applying the rules Swap_1 and Swap_2, the attributes of *student* and *course* will move with their owner object classes. The relationship attribute *grade* is thus attached to the object class *student* again. Applying the rule Swap_4, the object class *lecturer* will move up with *course* as a whole since *course* is the lowest participating object class of *dcl*. On the other hand, *tutor* will be attached to *student* because *student* is the lowest participating object class of *cst*. In this way, the semantics of the two relationship types are kept intact. Furthermore, the view obtained is the same as the original source schema in Figure 1. Thus, the view in Figure 3 is a reversible view because we can produce the original source schema back by applying swap operator on it.

4 Participation Constraints in Views

During the design of semistructured views, new relationship types may be derived from existing relationship types. The view may change the order of participating object classes of an existing relationship type. Thus, we need to determine the participation constraints of the relationship type in the view. We design four rules to handle the participation constraints for relationship types in semistructured views under the four view operators.

We use p and c to denote the parent and child participation constraints of an original relationship type R respectively. Likewise, we use p' and c' to denote the parent and child participation constraints of a derived relationship type R' .

The first rule handles the case when a *swap* operator is applied on two participating object classes of a binary relationship type. The order of the two participating object classes will be reversed in the view schema. Thus, in the new relationship type in the view, the participation constraints will also be reversed.

Rule PC_1: *If R' is derived in the view by swapping two participating object classes of an existing binary relationship type R in the source schema; then $p' = c$ and $c' = p$.*

Rule PC_2: *If R' is derived in the view by swapping two participating object classes in an existing n -ary ($n > 2$) relationship type R in the source schema, and O_1, O_2, \dots, O_n is participating object classes of R' in the order from ancestor to descendant in the view schema; then*

*for p' : If there exists a functional dependency $\{O_1, O_2, \dots, O_{n-1}\} \rightarrow O_n$, then set p' to be 1:1, otherwise set p' to be 0:n (or *).*

*for c' : if there exists a functional dependency: $O_n \rightarrow \{O_1, O_2, \dots, O_{n-1}\}$, then set c' to be 1:1, otherwise c' is set 0:n (or *).*

Rule PC_3: *If R' is derived in the view by projecting an existing relationship type R in the source schema, and O_1, O_2, \dots, O_n is participating object classes of R' in the order from ancestor to descendant in the view schema; then*

*for p' : If there exists a functional dependency $\{O_1, O_2, \dots, O_{n-1}\} \rightarrow O_n$, then set p' to be 1:1, otherwise set p' to be 0:n (or *).*

*for c' : if there exists a functional dependency: $O_n \rightarrow \{O_1, O_2, \dots, O_{n-1}\}$, then set c' to be 1:1, otherwise c' is set 0:n (or *).*

Rule PC_4: *If R' is derived in the view by joining one relationship type $R_1 (O_{11}, O_{12}, \dots, O_{1n})$ with another relationship type $R_2 (O_{21}, O_{22}, \dots, O_{2m})$, where $O_{1n} = O_{21}$ is the common object class they are joined on, then*

for p' : If there exists a functional dependency $\{O_{11}, O_{12}, \dots, O_{1(n-1)}, O_{22}, \dots, O_{2(m-1)}\} \rightarrow O_{2m}$, or a functional dependency $\{O_{22}, O_{23}, \dots, O_{2(m-1)}\} \rightarrow O_{2m}$, or the two functional dependencies $\{O_{11}, O_{12}, \dots, O_{1(n-1)}\} \rightarrow O_{1n}$ and $\{O_{21}, O_{22}, \dots, O_{2(m-1)}\} \rightarrow O_{2m}$; then set p' to be 1:1, otherwise, set p' to be 0:n.

*for c' : If there exists a functional dependency $O_{2m} \rightarrow \{O_{11}, O_{12}, \dots, O_{1(n-1)}, O_{22}, \dots, O_{2(m-1)}\}$, then set c' to be 1:1, otherwise set c' to be 0:n (or *).*

The second rule handles the case where the order of participating object classes of n -ary ($n > 2$) relationship types is changed. The third rule then handles the case where new relationship types are derived by projecting existing relationship types. Finally,

the last rule handles the case where new relationship types are derived by joining existing relationship types.

5 Conclusions

Existing systems for semistructured views do not maintain semantics at all in the process of designing the views. Thus, they cannot guarantee the validity and reversibility of the views. This paper proposes a novel approach to solve the two issues based on a semantically rich semistructured data model – ORA-SS. The *swap* operator is unique in semistructured data as it interchanges the positions of parent and child object classes, and raises the issue of view reversibility. In this paper, we have developed a complete set of rules for the *swap* operator and show that the proposed approach can maintain the evolution and integrity of relationships. We also examine the possible changes for participation constraints of the relationship types and propose rules to keep the participation constraints correct in the view. To the best of our knowledge, this approach is the first work to employ a semantic data model for maintaining semantics of semistructured views and solving the reversible view problem. By considering the validity and reversibility of semistructured views, this approach provides for a more robust view mechanism so that we can greatly exploit the potential of XML/semistructured data to exchange data on the Web.

References

1. Serge. Abiteboul, S. Cluet, L. Mignet, et. al., “Active views for electronic commerce”, VLDB, pp.138-149, 1999.
2. Chaitanya. Baru, A. Gupta, B. Ludaescher, et. al., “XML-Based Information Mediation with MIX”, ACM SIGMOD Demo, 1999.
3. Michael. Carey, J. Kiernan, J. hanmugasundaram, et. al., “XPERANTO: A Middleware for Publishing Object-Relational Data as XML Documents”, VLDB, pp. 646-648, 2000.
4. Michael. Carey, D. Florescu, Z. Ives, et. al., “XPERANTO: Publishing Object-Relational Data as XML”, WebDB Workshop, 2000.
5. Ya Bing. Chen, Tok Wang Ling, Mong Li Lee, “Designing Valid XML Views”, ER Conference, 2002
6. Ya Bing Chen, Tok Wang Ling, Mong Li Lee, “Automatic Generation of SQLX Definitions from ORA-SS Views”, DASFAA, 2004.
7. Sophie. Cluet, P. Veltri, D. Vodislav, “Views in a large scale xml repository”, VLDB, pp. 271-280, 2001.
8. Gillian. Dobbie, X.Y Wu, T.W Ling, 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.
9. Mary. Fernandez, W. Tan, D. Suciu, “Efficient Evaluation of XML Middleware Queries”, ACM SIGMOD, pp. 103-114, 2001.
10. Mary. Fernandez, W. Tan, D. Suciu, “SilkRoute: Trading Between Relations and XML”, World Wide Web Conference, 1999.
11. Philip. Bohannon, H. Korth, P. Narayan, S. Ganguly, and P. Shenoy. Optimizing view queries in ROLEX to support navigable tree results. VLDB, 2002.
12. Alin. Deutsch and V. Tannen. MARS: A System for Publishing XML from Mixed and Redundant Storage. VLDB, 2003.