On View Transformation Support for a Native XML DBMS

Daofeng Luo[1], Ting Chen[2], Tok Wang Ling[2], Xiaofeng Meng[1]

[1]Information School of Remin University of China
Beijing 100872, China
xfmeng@mail.ruc.edu.cn
[2] School of Computing, National University of Singapore
Lower Kent Ridge Road, Singapore 119260
{chent,lingtw} @comp.nus.edu.sg

Abstract. XML is becoming the standard data exchange format. View or transformation of XML data is important and frequent operation in XML data integration and publishing. In schema-based view transformation, users define view schema over sources to obtain view results. This declarative approach alleviates user from writing complex scripts to perform view transformation. Current available schema formats are unable to express views with complex semantic constraints. In this paper, we introduce a semantically expressive XML data model: Object-Relationship-Attribute model for Semi-Structured data (ORA-SS), which allows users to define view schemas with rich semantic meanings. Combine with ORA-SS, we use a native XML DBMS: OrientStore to perform accurate and efficient view transformation.

1 Introduction

Traditionally, view is an important aspect of data processing. In the context of XML, there are two main approaches to define views over source XML data. One way is to define views or queries in script languages like XQuery[7] or XSLT[8]. The alternative approach is to define views through source schema and view schema mappings. Systems like Clio[6] fall into this category. This declarative approach alleviates user from writing complex scripts to perform view transformations. Schema mappings can then be translated into XQuery (or XSLT) scripts or sequence of operations in back-end XML query processors. In this paper, we focus on view transformation through schema mapping.

View transformation via schema mapping needs an expressive XML schema representation. To see this, let us examine a sample XML document in Fig. 1(a). It contains information about researchers working under different projects and their publication lists (Note that the publication lists are not dependant on which project the researchers work in). To use simple tree/graph-structure schema languages for view (target) schemas (very similar to the one used in Clio) causes ambiguity. For example, the view schema in Fig. 1(c) can be interpreted in two different ways:

1. For each project, list all the papers published by project members; for each paper of the project, list all the authors of the paper.

2. For each project, list all the papers published by project members; for each paper of the project, list all the authors of the paper who are working for the project.

The different interpretations result in different view results. In this paper we introduce a XML schema representation: Object-Relationship-Attribute model for Semi-Structured data (ORA-SS) [2], which can clearly define the semantics of source data and views. We next show how to process view transformations defined by schema mapping efficiently on a native XML DBMS OrientStore [5].

This paper is organized as follows: Section 2 introduces our XML data model ORA-SS and our native XML DBMS: OrientStore. Section 3 presents the view transformation algorithm. Section 4 shows the performance of our view transformation algorithm. Section 5 concludes the paper.

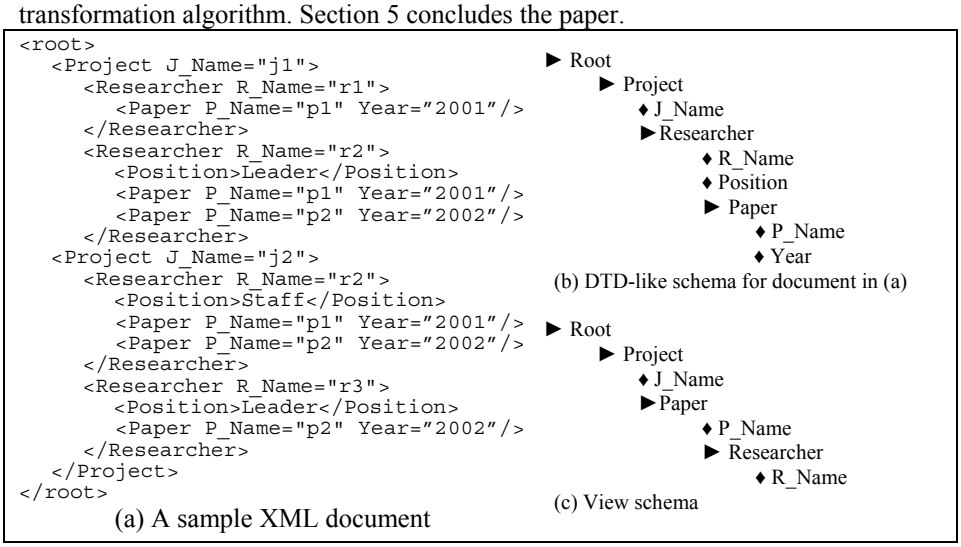


Fig. 1. A sample XML document with source and view schemas

2 System and Data Model for View Processing

In this section, we first introduce the conceptual data model used in view transformations. Next we give an overview over our XML DBMS: OrientStore.

2.1 ORA-SS (Object-Relationship-Attribute model for Semi-Structured data)

ORA-SS is a semantically expressive data model [2]. It has two important types of diagrams. An ORA-SS instance diagram represents a XML document while an ORA-SS schema diagram models the schema. An ORA-SS schema diagram has the following basic concepts: (1) Object Class (2) Relationship Type: Two or more object classes are connected via a relationship type in schema diagram. (3)Attribute:

Attributes are properties of an object class or a relationship type. For each object class, an ORA-SS schema indicates which relationship types it participates in. Similarly for each attribute, an ORA-SS schema explicitly indicates its owner object class or relationship type. These features are not present in DTD or XML Schema.

Let us examine an example. Figure 2(a) shows an ORA-SS schema diagram for the XML file in Fig. 1a. The ORA-SS schema diagram explicitly indicates the following facts about XML documents conforming to the schema: (1) There are two binary relationship types in the schema: *Project-Researcher* (JR) and *Researcher-Paper* (RP). A project can have several researchers and a researcher can work in different projects. Meanwhile, the set of papers under a researcher doesn't depend on the project he/she works in. (2) `Position` is an attribute of relationship type JR instead of `Researcher`. This means that a researcher may hold different positions across projects he works in. (3) `Year` is a single-valued attribute of object class `Paper`. Different occurrences of the same paper will always have the same `Year` value. (4) `J_Name,R_Name` and `P_Name` are identifiers of object classes `Project` , `Researcher` and `Paper` respectively, as indicated by solid circles.

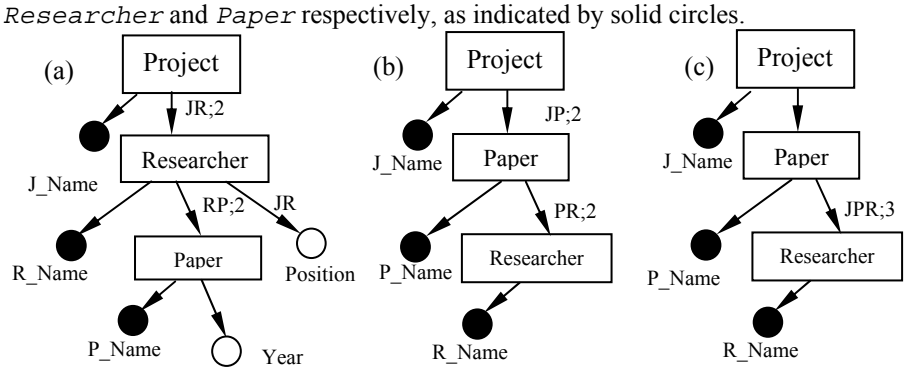


Fig. 2. (a) The ORA-SS schema diagram for the XML file in Fig 1(a). (b) and (c) Two different view schemas defined over the source schema in (a)

ORASS schema diagrams can be used to define views. Semantic meanings of views can be inferred from their ORA-SS view schemas. In Fig.2b and Fig.2c, although the two view schemas over source schema in Fig 2a look nearly identical, they represent quite different semantic meanings:

1. Fig. 2(b) has two binary relationship types. It is designed to find all the papers published by researchers in a project; and for each paper to find all of its authors.
2. Fig. 2(c) has only one ternary relationship type. The view is defined to find all the papers published by researchers in a project; however, for each paper Fig 2(c) only wants to finds authors working for the project.

In other words, the schema in Fig 2(b) requires that Project *j*, Paper *p* and Researcher *r* are on the same path in the view if *j* is on the same path *P* as *p* AND *p* is on the same path *P'* as *r* in the source data (*P* may not be same as *P'*), whereas Fig 2(c) requires *j, p* and *r* should locate on the same path in the source.

2.2 OrientStore

OrientStore is a native XML DBMS which targets for the storage and query processing of XML data. OrientStore uses *E*lement-*B*ased *C*lustering (*EBC*) storage strategy. *EBC* treats an element node together with its attribute nodes and text nodes (if any) as a single record. Element nodes (records) with the same tag name are clustered and organized as a list. This clustering method facilitates fast retrievals of elements of the same tag and structural join[1] of two element lists, both of which are important query operations. *EBC* gives numbers for nodes in XML document. Numbers for nodes in XML data tree can be calculated in the following manner: (1) The root element has number 1. (2) Perform a pre-order traversal (i.e. Document order) on the XML document. For node *x*:

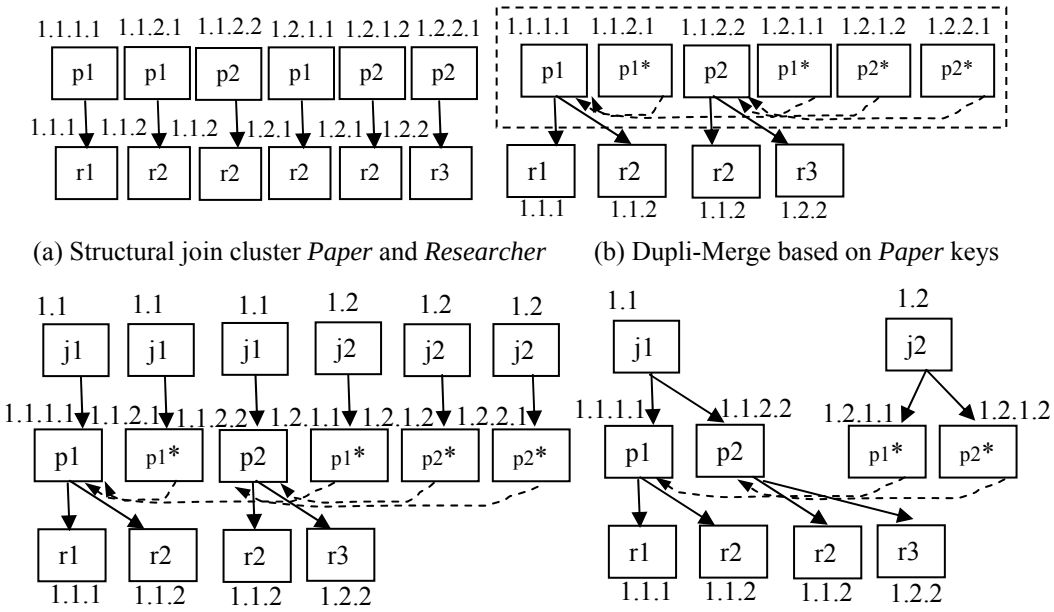
Number(*x*) = Number(*x*.parent) + "." + position of *x* in *x*.parent's child List

It is easy to see that the numbering scheme can tell if two nodes are located along the same path in constant time. As an example, for the XML file shown in Fig. 1(a), there are four clusters allocated based on the ORASS schema in Fig. 2(a): *Project*(with J_Name), *Researcher* (with R_Name), *Paper*(with P_Name and Year) , *Position*. The streams are shown as follows. The records in the streams are separated by semi-colons and the numbers for records are shown in the parentheses.

Project:	j1 (1.1); j2 (1.2)
Researcher:	r1 (1.1.1); r2(1.1.2); r2(1.2.1); r3(1.2.2)
Paper:	p1,2001(1.1.1.1); p1,2001(1.1.2.1); p2,2002(1.1.2.2); p1,2001(1.2.1.1); p1,2001(1.2.1.2); p2,2002(1.2.2.1)
Position :	Leader(1.1.2.3); Staff(1.2.1.3); Leader(1.2.2.2)

3. View Transformation

In this section, we present the highlight of our view transformation algorithm. We use a bottom-up view transformation algorithm. We use the view schema in Fig. 2(b) as the example. In our algorithm, the relationship Paper-Researcher in the view schema is constructed first. As Fig. 3(a) shows, we perform a structural join [1] on element clusters *Paper* and *Researcher*, with the result list sorted on object numbers of *Paper*. It should be noted that a merge at this stage is necessary because different occurrences of the same paper now have only partial author list. However, a normal merge (e.g. merge the three *p1* instances) will result in an unsorted *paper* list which can't be used in subsequent structural joins. Instead we keep the order of *paper* objects intact and just merge the *author* lists for each distinct *paper*. To avoid duplicates, we put the merged child list under the first occurrence ONLY and let other duplicated objects point to the first copy. Next we perform another structural join on cluster *project* and the sorted *paper* list. Because *project* has no more parent in the view schema, we just perform a normal merge after the structural join. We can perform a DFS on the result graph to get the XML output document. In essence, object numbers are used for structural joins and object identifiers are used in merge and dupliMerge phases. The number of rounds of structural joins and merge/dupliMerge is determined by the number of relationships in the view schema.



(a) Structural join cluster *Paper* and *Researcher* (b) Dupli-Merge based on *Paper* keys

(c) Structural join cluster *Paper* and *Project* (d) Merge based on *Project* keys

Fig. 3. Bottom-Up View Transformation for view schema Fig 2b. p1 ← ---- p1* means p1*'s children list pointer refers to p1's children list.

4. Experiments

In this section we present a summary of performance measurements of our view transformation algorithm. We compare our method with state-of-art XML query processing engines like SAXON[3] XSLT processor and Galax[4] XQuery processor. Our test platform is a PC with Pentium-3 867MHz CPU and 512 Mbytes RAM running Win XP. We synthesize the JRP (Project-Researcher-Paper) dataset.

Table 1. Performance summary of view transformation on JRP data set

View Schema	Source Document		View	Running Time(Second)		Ratio (Saxon/ OrientStore)
	Size(MB)	# of Objs	# of Objs	OrientStore	SAXON	
Fig 2b	40	270,000	600,000	42.1	-*	>250
	80	540,000	1200,000	84.6	-*	>250
Fig 2c	40	270,000	480,000	30.1	58.6	1.9
	80	540,000	960,000	67.4	122.4	1.8

* The running times of SAXON 7.5(as well as Galax XQuery processors) are too large to measure.

Our view transformation algorithm outperforms SAXON 7.5 in transformations for all view schemas. Most noticeably, our algorithm performs transformation for

view schemas having multiple relationship types on one path (view schema in Fig 2b) efficiently while the running time of SAXON is unacceptable even for small files. The reason is that for each paper of a project *j* a document-wide search for its complete author list is required because not all the authors work for the project *j*.

5 Conclusion

The starting point of this paper is the observation of the problems with two kinds of XML view transformation systems. High level systems like Clio, which perform view transformations via schema-mapping, have problems with defining views with complex semantics. On the other hand, general systems like XSLT and XQuery processors require user to write transformation scripts themselves. Our approach also uses schema-mapping like those high-level systems; however, it performs view transformation on a native XML DBMS: OrientStore. The use of ORA-SS as underlying schema representation allows us to express view schemas with a great variety of semantic meanings. At the same time, our system requires much less time than general XML processors in processing view transformations.

Acknowledgements

This research was partially supported by the grants from 863 High Technology Foundation of China under grant number 2002AA116030, the Natural Science Foundation of China(NSFC) under grant number 60073014.

Reference:

1. Shurug Al-Khalifa, H. V. Jagadish, Nick Kouda, Jignesh M. Patel, Divesh Srivastava, YuqingWu. Structural Joins: A Primitive for Efficient XML Query Pattern Matching. In *Proceedings of ICDE*, 2002
2. Gillian Dobbie, Wu Xiaoying, Tok Wang Ling, Mong Li Lee: ORA-SS: An Object-Relationship-Attribute Model for Semistructured Data TR21/00, Technical Report, Department of Computer Science, National University of Singapore, December 2000.
3. Michael Kay. SAXON XSLT Processor. http://saxon.sourceforge.net/
4. Galax. An XQuery Processor. http://db.bell-labs.com/galax/
5. Xiaofeng Meng, Daofeng Luo, Mong Li Lee, Jing An. OrientStore: A Schema Based Native XML Storage System. In *Proceedings of the 29th VLDB Conference, Berlin, Germany, 2003*
6. Lucian Popa, Mauricio A. Hern´andez ,Yannis Velegrakis, Ren´ee J. Miller, Felix Naumann, Howard Ho. Mapping XML and Relational Schemas with Clio. *In ICDE 2002 Demo*, 2002
7. XQuery. http://www.w3.org/XML/Query
8. XSLT 2.0. http://www.w3.org/Style/XSL