A Survey to View Update Problem

Haitao Chen and Husheng Liao

Abstract—**XML has become the de facto standard for representing and interchanging data in web-based applications. And XML view, a virtual window for specified users, has been widely applied. In practical system, users encounter the so-called view update problem when they need update source data through the view. For a long time, the view update problem is an open question in database community. With the development of various data models, the corresponding view update problem has been widely researched. In this paper, we introduce the conception of view update problem. We survey and compare previous approaches. Especially, we emphasize the role of semantics. Focusing on the problem in XML context, we give a discussion and propose a framework, which collects the semantic information at view definition time. Some related techniques for implementing this framework are further introduced.**

Index Terms—**database, semantics, view updates, XML**

I. Introduction

XML has become the de facto standard for representing and interchanging data in web-based applications. Major commercial database systems provide the ability to export relational data to materialized XML views. Since more and more XML documents have emerged, native XML database has been designed and developed. The World Wide Web Consortium publishes XQuery, which is a standardized language for combining documents, databases, Web pages and almost anything else. And the XML Query Working Group is developing an update facility for XQuery (XQuery Update Facility), this lets users write Query expressions that change documents and perhaps save the result.

From another point of view, wide application of XML makes managing and maintaining users' access inevitable. Various methods, such as access control, XML view, have been employed for resolving it. View is the virtual data for users, which can separate application logic from data source and provide application-specific data. View has been successful in relational database community. The ability to create views over XML source data, not only secures the source data, but also provides an application-specific view of the source data.

While providing view, the system should give users the ability to access view, including query and update. It is comparatively simple to query the view. View update, however, is an open problem, which means to translate an update on view to an update sequence on source data, i.e. to make view update persistent. The key problem is there is not a one-to-one correspondence between database state and view state. Existed researches have proven not only the organized structure of data, but also the context of application can affect the view update problem [1], that is, view update is a problem involving semantic information. Typically, data semantic information is expressed by defining various constraints. Obviously, view, which is defined only by query language (XQuery, SQL), do not contain sufficient semantic information.

In this paper, we first introduce the conception of view update problem. We then survey and compare previous researches. In particular we emphasize the importance of semantic information. Finally, we discuss the problem and show a framework to resolve it. This framework collects the semantic information at view definition time. Some related techniques for implementing this framework are further introduced.

The rest of this paper is organized as follows. Section 2 provides preliminary terminology and introduces the view update problem. Section 3 surveys and compare the previous work. Section 4 discusses the problem and presents the framework, with further introducing some related techniques. Section 5 concludes the paper with a brief outlook on the future work.

II. View Update Problem

In 1974, Codd first reports the view update problem [2]. Then the problem has been researched widely and deeply in database community. The definition of the problem can be found from the early literature [1] [3]. In this section, we introduce the preliminary terminology and notations in XML context. We then give the conception of view update problem.

A. *Preliminary Terminology and Notations*

XML document can be regarded as a node-labeled tree. An XML document often combines one or more XML Schema Documents, Which include type information and semantic constraints.

Let d be an XML document, X' be a set of XML Schema documents associate with d. We shall use the notation d←X' to denote d conforms to X'. This paper only gives notations, the detail can be found from corresponding specifications.

Definition 1: XML database is a set of XML Schema, which is denoted by X.

Manuscript received December 20, 2009. This work was supported by Beijing Municipal Natural Science Foundation (No. 4082003).

Haitao Chen is with College of Computer Sciences, Beijing University of Technology, Beijing, 100124 China (corresponding author to provide phone: 86-10-67392987; fax: 86-10-67391745; e-mail: chenheyuzhi@emails.bjut.edu.cn).

Husheng Liao is with College of Computer Sciences, Beijing University of Technology, Beijing, 100124 China.

Definition 2: A state of XML database X is a set of XML document instances, which is denoted by s. And $\forall d \in s$, $\exists X'(X' \subseteq X)$, $d \leftarrow X'$.

Definition 3: XML View is a XML database. And there is a mapping f, given a database state s, we can get a view state f(s), f is called view definition mapping. The set of view states is denoted by $f(S)=\{f(s) \mid s \in S\}$, S is the set of database states.

Definition 4: $u_s : S \rightarrow S$ denotes update operations on XML database. The set of update operations on XML database is denoted by U_s.

Definition 5: $u_f : f(S) \rightarrow f(S)$ denotes update operations on view. The set of update operations on view is denoted by U_f.

B. *The Definition of View Update Problem*

View updates are the update operations that occur on the view. Since the view is only an uninstantiated window onto the database, any updates specified against the database view must be translated into updates against the underlying database. We employ a classic figure to describe the problem more vivid.

Fig. 1 shows the process. The initial database state, *s*, is mapped by means of the view, f, into view state f(s). A view update $u \in U_f$ against this view state to obtain new view state u(f(s)). The view update, u, must be translated into a database update sequence to make update persistent. The database update sequence, denoted by T(u) and $T(u) \subseteq U_s$, is performed on the database state to obtain a new database state T(U)(s). This new database state is denoted by s'. By the same mapping f, we can obtain the view state f(s').

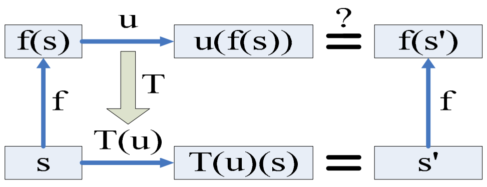

Figure 1. View Update

Definition 6: The above T is called update translator.

Definition 7: T has no side effects if u(f(s))=f(s').

View update problem is to find a translator that can translate view update reasonably under some conditions. Traditionally, T should have no side effects.

III. Previous Researches

With the development of various data models (relational data model, object-oriented data model, deductive data model, XML data model), view update problem has been studied widely and deeply. In particular, Researches in relational database community have far-reaching effect. We then survey previous researches on view update in terms of the methods they employ.

A. *Constant Complement*

"Constant Complement" is a classic theory about view update problem. Bancilhon and Spyratos [4] explain this theory in detail. Intuitively, the complement of view describes the information that can not be found in the view. The complement of one view definition mapping is another view definition mapping. The view constructed by these two mappings has the same amount of information as the view constructed by the identity mapping, i.e. the view constructed by these two mappings contains all information of source data. Given view definition mapping f and view update u, if g is the complement of f, there is only one view update translation under constant complement, and a theoretical formula is introduced in [4]. Since a view can have many different complements, the choice of a complement determines an update policy. Less complement permits more updates on view. Therefore, it is better to find a smaller complement.

Translating view update under constant complement is a well-known theoretical method for resolving view update problem, which has far-reaching effect on follow-up work, such as Hegner [5] [6] and Lechtenbörger [7]. In xml context, reference [8] tackles the XQuery view update problem by applying the constant complement theory, which focuses on the round-trip XML view update problem. The XML data is loaded into relational database. A view that is the same as the original xml data is extract from the relational database by XQuery. This view query is called extraction query, and this special view is called twin-view. Reference [8] shows the round-trip XML view is updatable in terms of constant complement theory and implements a decomposition-based update strategy in Rainbow system.

Although constant complement gives a pretty resolution to view update problem, there are still some inherent problems:

First of all, it makes a strict restriction. Reference [9] gives an example to show there is reasonable view update translation which is not translated under constant complement. Reference [10] can deal with a larger class of views, we review it later.

Secondly, it is difficult to compute the complement of view in practical application. There are lots of researches to study how to compute the complement efficiently [11-13].

Finally, this method is at schema level, which does not consider the data semantics and application semantics. However, as showed in [1], semantics is very important to resolve the ambiguity of translation.

B. *"Clean Source"*

Dayal and Bernstein [3] put forward the "clean source" theory to explore the conditions of translating view update reasonably. Source denotes some part of source data that is needed to construct the corresponding part of view data. A "clean source" is a source and fulfills the following condition: any other part of the view can obtain the source from the source data that do not contain the "clean source". In order to express the relationship between view data and source data, a view-trace graph is introduced in [3]. And a view-dependency graph is employed to describe the functional dependencies in data. As a whole, reference [3] explores syntactic translation procedures, and derives the conditions that are easy to check.

To deal with view update of xml views over relational databases, Braganholo et al [14-17] make use of "clean source" method. The key idea is to map XML views into relational views, then existing work on updating relational views can be employed to determine the updatability of the relational views, in particular, the "clean source" method is adopted.

Reference [18] studies the updatability of XML views that wrap the relational data. A "clean extended-source" is presented, which extends "clean source" by taking into account foreign keys. A view update is classified as either un-translatable, conditionally or unconditionally translatable. To classify a given update into one of the three categories, reference [18] also gives a graph-based algorithm, which is similar with [3]. Reference [19] is the follow-up work of [18], which improves on the previous algorithm and proves the algorithm by "clean extended-source" theory.

The "clean source" theory is a syntactic approach. But the assumptions and conditions make it too restricted. And it also has no sufficient semantics. Although reference [3] devises view-dependency graph for functional dependencies and [18] extends "clean source" by considering foreign keys, we think there are still much semantic information is significant for resolving the problem, such as application semantics, which is discussed by Keller [1] [20] [21].

C. *Consistent Views*

Gottlob et al [10] considers consistent view, which has the following property: If the effect of a view update program on a view state is determined, the effect of the corresponding database update is unambiguously determined. View consistency is an important property, which makes database systems (with views) accessible to formal specification and verification techniques. And consistent views have interesting properties with respect to the concurrent execution of transactions. It is shown that if concurrent transactions on the view are translated to the database level, then properties such as serializability and noninterference are preserved.

It is also shown that the class of views treated by constant complement is a subclass of the consistent views. In fact, the method in [10] is a generalization of [4]. To deal with the much larger class of consistent views, the notion of complement is still used. However, the complement does not need to remain invariant. The complement is allowed to decrease according to suitable partial order instead.

Although the class of consistent views is an important super class of the views discussed in [4], there are reasonable views that are not consistent. It also lacks of adequate efficient algorithms to make the method more practical. And more importantly, it only considers the syntactic effects.

D. *Collecting Semantic Information at View Definition or Update Time*

Masunaga [22] presents a semantic approach to design a view update translator for relational database systems. Masunaga [22] argues that because of the ambiguity of translation, semantic information is necessary to solve it. But the semantics captured at view definition time is not sufficient, and it may involve the end-users at runtime, i.e. at view update time.

Keller [20] presents the problem of view update in relational context, and the view involves selections, projections and joins. Firstly, five criteria are given to describe the translations that are regard as reasonable. Then all translators satisfied these criteria are enumerated. The main novelty is Keller shows an insert or delete operation on the view may be translated into modify operation in specific application context. Keller argues extra semantics is needed to choose from all translators. Reference [1] and [21] present the role of semantics in translating view updates in detail. Keller proposes a resolution, which collecting semantic information at view definition time. That is to interact with DBA and obtain necessary semantics. Then the semantics can make a choice among translators that enumerated by [20].

Larson and Sheth [23] is influenced by [22] and [20], which makes use of syntactic and semantic knowledge to choose from different translators. It concludes the categories of knowledge that is useful in dealing with view update problem. It interacts with the DBA at view definition time to capture semantics. It also interacts with the end-user at view update time to capture additional application semantics and database semantics.

Choi et al [24] explores the view update problem for XML views published from relational data. Because there are subtree instances that can be shared, reference [24] focused on the side effects that are brought by view updates. The difference from previous works is it does not reject the side effects, but consults with end-users to obtain a more reasonable result. That is, it allows end-users to determine whether or not the side effects are reasonable. It can be regarded as capturing necessary semantics at view update time.

Lots of practical examples have showed the importance of semantics to view update problem. All of the above works have taken note of it and attempted to collect semantics at view definition or update time. We believe that it is vital to take into account semantics. And both database semantics and application semantics are required. Some semantics can be captured at view definition time, but the remaining semantics have to be collected at view update time. It is acceptable to interact with DBA at view definition time, but it burdens end-users to consult with them at view update time. It also conflicts with the original intention of view. Thus, it is important to maximize semantic information that can be gathered at view definition time and minimize semantic information that have to be collected at view update time, which deserves to be studied further.

E. *Identity Preservation*

In object-oriented database, objects have an identity. If the query language can preserve objects' identities, view object can find the corresponding source object by its identity. Therefore, view update problem becomes simple in this context at syntactic level. Scholl et al [25] presents this key idea and introduces object preserving operator semantics. Query language, however, usually contains identity-updating operator, which restricts this method in practical applications. And it also ignores the role of semantics in translating view update.

It deserves special attention in XML context. XQuery Data Model specifies nodes of XML document have node identity, which is similar to object identity. Thus, node identity can be used to build the correspondence between view node and source data node. And most of XQuery expressions are identity-preserving.

F. V-R Model

Pan and Yang [26] proposes an object model at the conceptual level, which is called the view class and real-world object model (V-R model). The major goal of the V-R model is to support updatable views efficiently for the object-oriented database systems. In the real world, people always observe an entity form different viewpoints. In the V-R model, people, viewpoints, and entities correspond to users, view classes, and real-world objects, respectively. Then we review the formal conception in detail.

A real world object (RO) is an entity, either concrete or logical. A RO is represented as a tuple <I, D>, where I denotes the unique real world object identifier (RID), D denotes the stored data. A view class C is represented as a tuple <S,N,T,E>, where S denotes the superclass of C, N denotes the class name, T denotes the intension, which is a set of attributes, E denotes the extension, which is a set of conceptual objects (CO). A CO is represented as a tuple <r, vf>, where r is an RID, vf is the view function. A view C is represented as a tuple <N,T,Q,E>, where N denotes the view name, T denotes the intension, Q is the instance population, which is a query, and E denotes the extension, which is a set of conceptual objects.

The V-R model is closed against basic query operations (Restriction, Projection, and Union, etc). There always exist a one-to-one mapping between each CO of the view and a corresponding set of Cos of operands on which the view is defined. This property makes the view updatable in syntactic level.

To clarify the semantics of view class updates, object evolution is introduced, which means an RO may change the amount of its stored data. The fundamental semantics for view updates is to establish or to break the association between the ROs in a CO. Base on above semantics, the view update translation is proposed.

The major idea of this approach is to use object identifier and keep the mapping pairs of operands' Cos in the corresponding CO of the resulting view. However, it is also a syntactic level method. When there are ambiguities, to choose a reasonable translation is the users' responsibility, which is similar to [23]. And the V-R model is tightly coupled with object-oriented data model, which makes it difficult to apply to other models.

G. Data Abstraction

Paolini and Pelagatti [27] thinks database as an abstract object, users can access this object by a set of operations. Reference [28] and [29] model database and view as data abstraction. Then many researches employ data abstraction to deal with view update problem [30-32].

Kozankiewicz [33] explores the updatable XML view. The key idea is to contain the information about intents of updates in view definitions. View definitions are regarded as complex entities, which is similar to the spirit of data abstraction. The information content of virtual objects is defined together with all the required view updating operations. That is, all update operations that are allowed on the view are specified by view definition.

This approach combines specific procedures with the view. These procedures show how to access the source data. It can be considered as capturing the semantic information at view definition time. It describes how to perform updates, and reflects the users' real intents. But it is awkward since it can not be changed with different contexts.

H. Constraint Satisfaction

In order to obtain a reasonable translation, Shu [34] shows the information is needed to describe the expected updates on the base relations. There may be syntactic information, which explains how to extract the view from the base relations. Also，There may be semantic information, which specifies integrity constraints and application requirements. In particular, semantic information can be collected at view definition or view update time. But how many constraints are adequate has no answer so far. Therefore, it is useful if constraints can be incrementally added.

A novel approach is presented in [34], which transforms a view update problem into constraint satisfaction problems (CSPs) by using conditional tables to represent relational views. Given a view and a view-update request, the CSPs can be constructed. It separates problem formalization and problem solving. The constraints can be added incrementally to restrict the number of solutions to the CSPs without the underlying constraint satisfaction algorithm being changed. At the same time, it is possible to apply the rich results of the CSP research to more efficiently solve the view update problem. It also shows how to treat with view modification and null value.

This approach is dependent of the instances of the database and the view and is to be applied at view update time. It can be a complement of other approaches. The critical problem is what constraints should be captured and how to capture them.

I. Bi-directional Transformation

Breenwald et al [35] introduces the idea of bi-directional tree transformations, which is implemented in a universal synchronization framework for tree-structured data. View has an important role in this framework: in order to synchronize disparate data formats, a single common abstract view is defined and a set of lenses that transform each concrete format into the abstract view are proposed. Lens is a vivid name, which is a pair of functions. One is called get, the other is called put. Intuitively, get denotes mapping concrete data into abstract data, and put denotes mapping the updated abstract data and the original concrete data into new concrete data. Reference [35] proposes two laws, PUTGET and GETPUT, to make sure the put function is injective. That is, abstract view and the original concrete view can determine one and only the current concrete view state. This disambiguates the translation of view update. Lenses are called well-behaved lenses, if they satisfy the above two laws. And very-well-behaved lenses are well-behave lenses that satisfy another law, PUTPUT. Interestingly, Pierce et al [36] shows the set of very-well-behaved lenses is isomorphic to the set of translators under constant complement [4], and the set of well-behaved lenses is isomorphic to the set of dynamic views [10]. Foster et al [37] introduces the design of lenses and how to resolve the view update problem of tree-structured data by bi-directional transformation.

The University of Tokyo also studies this topic intensively [38-41]. Reference [38] presents a point-free functional language, Inv, in which all functions definable are injective. This language, with relational semantics, has the same computational power with Bennett's reversible Turning machines. It is shown that for all possibly non-injective functions f: A→B, we can automatically derive an injective function f': A→(B, H), where H records the history of computation. Therefore, every program described by Inv is trivially invertible. Reference [39] extends Inv and develops a formalization of bi-directional updating that can deal with duplication and structural changes like insertion and deletion. For making tree transformations bidirectional, reference [40] proposes a new general transformation, called bidirectionalization. For a case study, it is shown any tree transformation in HaXML can be made bidirectional. Reference [41] develops an XML data update system which can update remote XML data through XML views. This architecture combines view updating and standard Web service technology. And it makes full use of bidirectional transformation to implement view updating.

Bi-directional transformation is a linguistic method, which has attracted wide attention in programming and database community. The key idea of bi-directional transformation stems from Reversible Computing, which reduces "heat" dissipation. It is sound to apply bi-directional transformation to resolve view update problem. The keystone of this method is to design a bi-directional transformation language, which can describe the relation between view and source data. This language should also include some necessary information, such as constraints. It can be thought inclusion of semantics in the design of language. Semantics, however, should be studied further when this method is applied.

J. Conclusion of previous methods

The difficult of view update problem is how to obtain a reasonable translator. We think that "reasonable" means the translation of view update can reflect users' intents. Thus, we believe semantics is necessary to resolve the problem. As shown in table 1, we make a summarization of these nine methods, in particular, we focus on semantic features.

In Table 1, we compare the above methods. To save space, we show the first three methods in the same column, since they have the similar properties. "Level" means the level at which the method resolves the problem, in particular, syntactic or semantic. We think abstract data and bi-directional transformation are at semantic level, since they contain semantics in abstract data type and bi-directional transformation language respectively. "Means" denotes the methods they employ. "Emphasize semantics" presents whether or not the method emphasizes the importance of semantics. "How to provide semantic information" shows how to capture the semantics.

We introduce a taxonomy of semantics that may be useful when tackling view update problem. As shown in Fig. 2, Semantic information includes data semantics and application semantics.

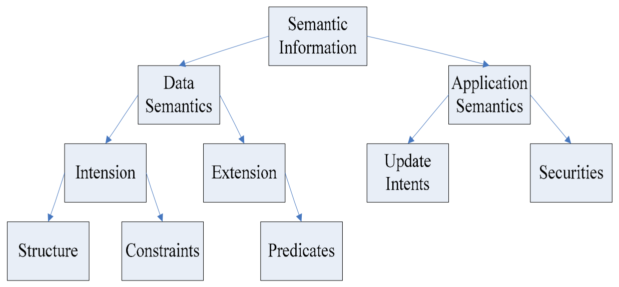

Figure 2. Taxonomy of Semantics Useful When Tackling View Update Problem

Data semantics contains intension and extension. Intension includes structure and constraints. Structure represents the semantic containing in the relation between data, such as order of data. Constraints express data semantic constraints (e.g., functional dependencies, integrity constraints). Extension shows what data are valid. Typically, it is characterized by some predicates. Application Semantics captures the intents of data from some aspects when we use them. As explained in [1], an insert statement on view may be translated into a modify statement on source data. Thus, we need update intents to instruct the translation. And securities tell us how to use the data in security. Typically, it may contain access control policies.

IV. Discussion and A Framework

A. Discussion

1) Update Semantics of Views

To resolve view update problem, it is necessary to know what the reasonable translation is, that is, what the update semantics of views is. There have been lots of researches about it in relational community. Traditionally, it is believed that translation that is consistent and has no side effects is reasonable. A stricter semantics is enunciated in [4], which requires all information missing from the view must be included in the complement. However, if there are some constraints on view data, e.g. functional dependencies A→B, side effects may be inevitable in order to maintain data consistency when A is modified. Thus, the updates on those data have to be rejected. But is it the users' intents? It need capture the real users' intents, which may expect to perform updates, and the side effects are tackled by system automatically. Therefore, to reject side effects is unilateral.

2) The Importance of Data and Application Semantics

Data semantics and application semantics are also useful for considering this problem [1] [23]. Data semantics is critical to describe the relationship between data, which can instruct how to use and maintain data reasonably. For example, we have to maintain integrity constraints when we update the view or the database. And many researches assume the update operation always is translated into the same kind of update operation, e.g. an insert operation is translated into another insert operation, which is call update monotony. However, even updating the same view with the same operation, the update monotony may not be satisfied, because an insert or delete operation against a view may be translated into a modify operation against the source data, which depends on the application context. It is necessary to capture the application semantics to resolve ambiguity.

3) No One-to-one Mapping

As presented above, the major problem is there is no one-to-one mapping between update on the view and sequence of updates on the source data. Under given view update semantics, for update on the view, there are three cases: a) there is no translation; b) there is only one translation; c) there are many translations. Obviously, the second case is expected result.

For the first case, the simplest way is to reject the update. Another way is to restrict the view or the updates on the view to avoid these invalid updates. For instance, we can provide a set of primitives for manipulating views, which can guarantee there always exists the valid translation. Similarly, we also can provide a set of operations for view construction to make sure no invalid update can occur against the view.

For the third cases, we can provide additional rules or constraints to limit valid translations, e.g. we require the translation produces produce the shortest update sequence. Also, we can add order information to all possible translations to obtain an optimal one, that is, a partial ordering relation may be introduced. Finally, we can interact with users to choose a translation that can reflect users' intents exactly.

4) Time for Collecting Semantic Information

As shown in Fig. 2, there are many kinds of semantic information. As emphasized above, Semantic information is very important to resolve ambiguities, which is the inherent property of view update problem.

Some semantic information, such as integrity constraints, can be captured at view definition time. Typically, DBA can use a tool to define a view, which is not only a stored query, but also some schema information.

But the semantic information collected at view definition time may be insufficient [22]. System can interact with users at view update time to obtain semantic information [22] [23]. Obviously, capturing semantics at run-time can find users' real intents, but it burdens end-users. And exposing these details is in contradiction to the original intents of the view.

5) The Role of Constraints

The constraints on data (both view and source data) make it indispensable to maintain consistency when the data is updated. In the context of XML, How to specify various semantic constraints has been studied intensively recently. In particular, as summarized in Fig. 3, XML data semantic constraints mainly include: unique, key, foreign key, functional dependency, inclusion dependency, inversion constraint and multi-value dependency. As relational database, these constraints play a critical role in normalization and maintaining consistency. XML Schema, a standard schema language propose by W3C, has mechanism to support unique, key and foreign key. Since XML is hierarchical nested structure, how to define traditional data semantic constraints is not a trivial problem. To satisfy various data semantic constraints and maintain data consistency make performing view update more complicated.

$$\begin{matrix}\text{Data Semantic}\\ \text{Constraint s}\\ \text{for XML}\end{matrix}\begin{cases}\text{Unique}\\ \text{Key}\\ \text{Foreign Key}\\ \text{Functional Dependency}\\ \text{Inclusion Dependency}\\ \text{Inversion Constraint}\\ \text{Multi} - \text{value Dependency}\end{cases}$$

Figure 3. Data Semantic Constraints for XML

6) Traditional Database or XML Database

XML is often used as a representation of other data formats. Therefore, XML view update, typically, is to translate update on XML view published over relational database. The simple approach is to transform the problem into traditional context. Then previous researches can be used. It seems reasonable, since the differences between two data models are resolved naturally.

However, with the development of native XML database, translating update on XML view published over XML source data is certain to be an important research area. Related techniques of XML are developing. Thus, it is difficult to study view update problem in XML context. For example, XML Schema can not describe functional dependency and there is no standard and mature XML update language (XQuery Update Facility is developing and may be a standard XML update language). But there are also some features of XML that may be useful when we study view update problem. For example, node identity gives the chance to track the data between view and database. Document order also provides the information of relative position between nodes.

B. A Framework and Critical Techniques

1) A Framework for Solving View Updates

We believe semantic information is vital. So we need a mechanism to describe necessary information. To extend XML Schema serves this target. We can add components to XML Schema to describe data semantics and application semantics. Then we can capture semantic information at view definition time by interacting with DBA. The information is stored by extended XML Schema. Under assistance of semantics, an algorithm is applied to translate update on view into a sequence of updates on source data. To facilitate view definition, we employ a conceptual schema to show the data we tackle, which can show the data in graphical notations and make it easier to use. Notably, we only collect semantic information at view definition time. It may be more reasonable to capture some semantic information at view update time. However, we should endeavor to avoid burdening end-users.

As shown in Fig. 4, the framework consists of view building client, view access client, view building/access server and data source. View building/access server includes model extractor, view definition analyzer, view update manager and view query processor. View building client obtains the conceptual schema of source data by model extractor. Then DBA can build expected view by manipulating conceptual schema. View definition analyzer translates view definition in conceptual schema format into extended XML Schema and XQuery statements. End-users obtain the view data by invoking view query processor. When end-users update view data, view update manager analyzes the updatability and perform the valid and expected translation if there is. The model to describe conceptual schema of the view, the strategies discussed above and the view update algorithm employed in framework determine the practicability and efficiency of the framework.

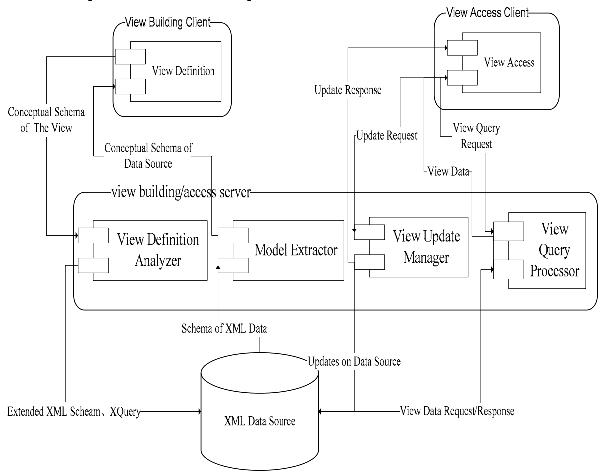

Figure 4. A Framework for Resolving View Update Problem

2) *Some Related Techniques*

a) Constraints for XML

Constraints, especially the integrity constraint, play a fundamental role in database design. With the development of native XML database, How to define constraints for XML has been studied widely. For the framework, we need to store constraints at logical level, which, typically, is implemented by a language. As shown in Fig. 3, various kinds of constraints can be specified for XML. However, there is no standard method.

XML Schema supports unique, key and foreign key. The constraints are declared in the element, which is the scope of the constraints. A selector element is used to choose the nodes that are affected by the constants. Properties involved in the constraints are denoted by one or more field element. A subset of XPath is used for locating elements and attributes. That means we can call the approach adopted by XML Schema is based on path.

Functional dependencies have been relatively more intensively studied than other constraints. Most of them are based on path, either absolute paths or more general paths [42]. There are also researches for other constraints [43] [44]. We do not go into detail here.

Since XML Schema is employed to record the constraints, we extend the syntax of XML Schema to support functional dependencies, multi-value dependencies etc, which is similar to the existed mechanism. However, informally speaking, a recursive style definition is introduced to emphasize the layers of different elements and attributes that are present in the constraint. And we believe the definition of value equality in [42] is not reasonable, since the content model of XML Schema has rich expressive power, especially the semantics of choice group. Therefore, we propose a new definition of value equality, which can capture the semantics of sequence group and choice group, even nested choice and sequence group. The algorithms for checking value equality and the satisfaction of constraints have been implemented by extending XML Schema Validator, which is supported by W3C. It is beyond the scope of this paper to present the details.

b) XML Conceptual Modeling

XML Schema can be regarded as the logical schema of XML model. We also need a conceptual schema of XML model to facilitate the users (DBA) to define views. To construct the view at conceptual level make the view definition can be achieved more efficiently and shared with other people.

Reference [45] proposes a methodology for defining view, which has three different levels of abstraction, that is, conceptual level, logical or schema level, and document or instance level. Our Framework accords with this methodology. The conceptual level can be modeled using UML, XSemantic Net [46], EER, etc. We may need to extend some well-established modeling language to serve XML conceptual modeling. There are also some new languages for XML conceptual modeling, e.g. XTM [47], ORA-SS [48].

In our framework, we employ a new modeling language that is developing by us. XML data sources are translated into the instances of this model. The users can manipulate these instances by a set of well-defined operations to construct the view. The model provides the mechanism to record semantic information.

c) How to Use Semantic Information

When the semantic information is collected at view definition time, we can use them to instruct view update translation. The semantic information can be classified into two kinds: one is the data semantic constraints, the other is the application semantics, which determines whether one update should be translated into another the same kind update. When the data are updated, we should check the application semantics to make sure whether to preserve the kind of update. Then we should check whether the involved data are constrained by data semantic constraints. If they are, we should update related data automatically to maintain data consistency, which produces update side-effects.

V. Conclusion

This paper introduces the view update problem. We survey and compare previous researches on view update in terms of the methods they employ. In particular, we emphasize the importance of semantics. After making a brief analysis, we present a framework for resolving the problem, which captures semantics at view definition time and is developing by us. For implementing this framework, we also further introduce some related techniques for implementing the framework.

We believe not only relation between view and source data but also semantic information, such as data semantics and application semantics, is necessary to solve the problem. The nature of the problem is the ambiguities arousing by translating view update. Additional semantics is needed to eliminate these ambiguities. Semantic information can be collected at view definition and view update time. We should maximize the information collected at view definition time to avoid burdening end-users.

Because XML and its related techniques are developing, it is a little difficult to consider this problem in XML context. However, as we surveyed, there are lots of researches on this topic, most of them employ old methods, which are developed in relational context. And some new ideas are proposed, such as bi-directional transformation. And special features of XML may be vital and useful.

How to describe, capture and use semantic information, how to deal with side effects are worthy of being studied further. And we are currently working on developing this framework.

REFERENCES

[1] A. M. Keller. The role of semantics in translating view updates. IEEE Transactions on Computers, v.19 n.1, pp.63-73, Jan. 1986.

[2] E.F. Codd. Recent investigations in relational data base systems. IFIP Congress, 1974, pp.1017-1021.

[3] U Dayal, PA Bernstein. On the correct translation of update operations on relational views. ACM Transactions on Database System, v.7 n.3, pp.381-416, 1982.

[4] F Bancilhon, N Spyratos. Update semantics of relational views. ACM Transactions on Database System, v.6 n4, pp.557-575, 1981.

[5] S. J. Hegner. Foundations of canonical update support for closed database views. Proceedings of the third international conference on database theory on Database theory, pp.422-436, November 1990, Paris, France.

[6] S. J. Hegner. An order-based theory of updates for closed database views. Annals of Mathematics and Artificial Intelligence, v.40 n.1-2, pp.63-125, January 2004.

[7] J. Lechtenbörger. The impact of the constant complement approach towards view updating. Proceedings of the twenty-second ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems, pp.49-55, June 09-11, 2003, San Diego, California.

[8] L. Wang, M. Mulchandani and E.A. Rundensteiner. Updating XQuery views published over relational data: a roundtrip case study. In XML Database Symposium, 2003, pp. 223–237.

[9] A. M. Keller. Comment on Bancilhon and Spyratos' "Update semantics and relational views". ACM Transactions on Database Systems (TODS), v.12 n.3, pp.521-523, Sept. 1987.

[10] G. Gottlob, P. Paolini and R. Zicari. Properties and update semantics of consistent views. ACM Transactions on Database Systems (TODS), v.13 n.4, pp.486-524, Dec. 1988.

[11] S. S. Cosmadakis , C. H. Papadimitriou. Updates of relational views. Journal of the ACM (JACM), v.31 n.4, pp.742-760, Oct. 1984.

[12] D. Laurent, J. Lechtenbörger, N. Spyratos, G. Vossen. Monotonic complements for independent data warehouses. The VLDB Journal — The International Journal on Very Large Data Bases, v.10 n.4, pp.295-315, December 2001.

[13] J. Lechtenbörger, G. Vossen. On the computation of relational view complements. In: Proceedings of PODS 2003, pp 142–149, Madison, Wisconsin, USA.

[14] V.P. Braganholo, S.B. Davidson and C.A. Heuser. Reasoning about the updatability of XML views over relational databases. Technical Report MS-CIS-03-13, Department of Computer and Information Science, University of Pennsylvania, 2003.

[15] V.P. Braganholo, S.B. Davidson and C.A. Heuser. Using XQuery to build updatable XML views over relational databases. Technical Report MS-CIS-03-18, Department of Computer and Information Science, University of Pennsylvania, 2003.

[16] V.P. Braganholo, S.B. Davidson and C.A. Heuser. From XML view updates to relational view updates: old solution to a new problem. In VLDB, pp. 276–287, 2004, Toronto, Canada.

[17] V.P. Braganholo, S.B. Davidson and C.A. Heuser. PATAXÓ: A framework to allow updates through XML views. ACM Transactions on Database Systems (TODS), v.31 n.3, pp.839-886, September 2006.

[18] L. Wang, E.A. Rundensteiner. On the updatability of XQuery views published over relational data. LNCS volume: 3288, pp.795-809, 2004.

[19] L. Wang, E.A. Rundensteiner and M Mani. Updating XML views published over relational databases: towards the existence of a correct update mapping. DKE Journal volume: 58, issue: 3, pp.263-298, 2006.

[20] A. M. Keller. Algorithms for translating view updates to database updates for views involving selections, projections and Joins. 4th PODS, ACM, March 1985.

[21] A. M. Keller. Choosing a view update translator by dialog at view definition time. Proceedings of the 12th International Conference on Very Large Data Bases, pp.467-474, August 25-28, 1986.

[22] Y. Masunaga. A relational database view update translation mechanism. Proceedings of the 10th International Conference on Very Large Data Bases, pp.309-320, August 27-31, 1984.

[23] A. Sheth, J. Larson and E. Watkins. TAILOR, A tool for updating views. LNCS v.303, pp.190-213, 1988.

[24] B. Choi, G. Cong, W. Fan, S.D. Viglas. Updating recursive XML views of relations. JCST 23(4): pp. 516-537 July 2008.

[25] M. H. Scholl, C. Laasch, C and M. Tresch. Updatable views in object-oriented databases, in PTOC.2nd DOOD Conf., Germany, Dec. 1991.

[26] Wen-Wei Pan,Wei-Pang Yang. An object model at conceptual level to support updatable views on object-oriented databases. Information Sciences,Volume 95, Issues 1-2, November 1996, Pages 29-48.

[27] P. Paolini, G. Pelagatti. Formal definition of mappings in a database. Proceedings of the 1977 ACM SIGMOD international conference on Management of data, August 03-05, 1977, Toronto, Ontario, Canada.

[28] P. Paolini. Verification of views and application programs. Proc. Workshop on Formal Bases for Databases, Toulouse (1979).

[29] P. Paolini, R. Zicari. Properties of views and their implementation. Advances in Data Base Theory 1982: pp. 353-389.

[30] E. K. Clemons: An external schema facility to support data base update. JCDKB 1978: pp. 371-.

[31] K. C. Sevcik, A. L. Furtado, Complete and compatible sets of update operators, Proc. Int. Conf. on Management of Data, ACM, June 1978.

[32] L. A. Rowe, K. A. Shoens, Data abstraction, views and updates in RIGEL, Proceedings of the 1979 ACM SIGMOD international conference on Management of data, May 30-June 01, 1979, Boston, Massachusetts.

[33] H. Kozankiewicz, J. Leszczylowski and K. Subieta. Updatable XML views. LNCS v.2798, pp.381–399, 2003.

[34] Hua Shu. Using constraint satisfaction for view update. Journal of Intelligent Information Systems, v.15 n.2, p.147-173, Sept./Oct. 2000.

[35] M. B. Greenwald, J.T. Moore, B.C. Pierce, A. Schmitt. A language for bi-Directional tree transformations. In the Workshop on Programming Language Technologies for XML (PLAN-X), 2004.

[36] C. Pierce and A. Schmitt. Lenses and view update translation. Manuscript, 2003.

[37] J.N. Foster, M.B. Greenwald and J.T. Moore, B.C. Pierce, A. Schmitt. Combinators for bidirectional tree transformations: a linguistic approach to the view update problem. ACM Transactions on Programming Languages and Systems. v.29 n.3, 2007.

[38] S.C. Mu, Z. Hu, and M. Takeichi. An injective language for reversible computation. In Seventh International Conference on Mathematics of Program Construction. Springer-Verlag, July 2004.

[39] S.C. Mu, Z. Hu, and M. Takeichi. An algebraic approach to bi-directional updating. In ASIAN Symposium on Programming Languages and Systems (APLAS), Nov. 2004.

[40] Zhenjiang Hu, K. Emoto, Shin-cheng Mu, M. Takeichi. Bidirectionalizing tree transformations. International Workshop on New Approaches to Software Construction (WNASC 2004), The University of Tokyo, Komaba, Tokyo, Japan, September 13-14, 2004. pp.3-22.

[41] Y. Hayashi, Dongxi Liu and K. Emoto et al. A web service architecture for bidirectional XML updating. LNCS v.4505, pp.721-732, 2007.

[42] JH Wang. A comparative study of functional dependencies for XML. 7th Asia-Pacific Web Conference, LNCS v.3399, pp. 308-319, 2005.

[43] Buneman P, Fan WF, Simeon J, et al: Constraints for semistructured data and XML. SIGMOD RECORD Volume: 30, Issue: 1, pp.47-54, MAR 2001.

[44] Vincent MW, Liu JX: Multivalued dependencies in XML. NEW HORIZONS IN INFORMATION MANAGEMENT LNCS v.2712, pp. 4-18,2003.

TABLE 1. COMPARISON OF THE ABOVE METHODS

	Constant Complement/ "Clean Source"/ Consistent Views	Collecting Semantic Information	Identity Preservation	V-R Model	Abstract Data	Constraint Satisfaction	Bi-directional Transformation
Level	Syntactic	Semantic	Syntactic	Syntactic	Semantic	Semantics	Semantic
Means	Translation under constant complement/ Clean Source Theory/ Generalization of constant complement	Capture semantics at view definition or update time	Build correspondence between source and view data by identity	A Conceptual Model	Regard view as abstract data	transforms a view update problem into CSPs	Bi-directional transformation language
Emphasize semantics	No	Yes	No	No	No	Yes	No
How to provide semantic information	No	Interact with DBA or user at view definition or update time	No	No	Combine specific procedures with view	Regarded semantic information as Constraints	The semantics of language