

Semantic Integrity Constraint Checking for Multiple XML Databases

Praveen Madiraju, Rajshekhar Sunderraman

Department of Computer Science
Georgia State University
Atlanta GA 30302

{cscpnmx,raj}@cs.gsu.edu

Shamkant B. Navathe

College of Computing
Georgia Institute of Technology
Atlanta GA 30332

sham@cc.gatech.edu

ABSTRACT

Global semantic integrity constraints ensure integrity and consistency of data spanning multiple databases. In this paper, we take the initial steps towards representing global semantic integrity constraints for XML databases. We also provide a general framework for checking global semantic integrity constraints in an XML setting. The general framework, we set forth is efficient for two reasons: 1) constraint check is carried out before updating the database; hence, we avoid any potential problems associated with rollbacks and 2) sub constraint checks are executed in parallel.

Keywords: XML Databases, Global XML Constraints, Constraint Checking in XML, XML Middleware

1. INTRODUCTION

Integrity constraints preserve the consistency, integrity of data and its checking has been widely recognized as an important area of research in deductive, relational and now in XML databases. Recent research results have been reported in [1], [3] and [6] considering the issue of validating key constraints for XML databases. To our knowledge, we have not seen any research on integrity constraint checking for multiple XML databases. We propose an approach that would efficiently and speedily check for constraint violations affecting multiple XML databases.

A single update (XUpdate [11]) on one site might cause a global constraint (*global XConstraint*¹) to be violated. In our constraint checking approach, constraint violations are checked at compile time, *before* updating the database. Our approach centers on the design of the *XConstraint Checker*. Given an XUpdate [11] statement and a list of global XConstraints, we generate *sub XConstraint*² checks corresponding to local sites. The results gathered from the sub XConstraints determine if the XUpdate statement violates any global XConstraints. Our approach is *efficient*, since we do not require the update statement to be executed before the constraint check is carried out and hence, we avoid any rollback situations. Our approach achieves *speed* as the sub constraint checks are executed in parallel.

1.1 Overview of the system

Figure 1 gives the overview of the system. We propose a three tier architecture. The server side consists of two or more sites hosting native XML database. In Figure 1, we show three sites S_1 , S_2 and S_3 . The client makes a XUpdate request through the middleware. The middleware consists of the *XConstraint Checker* and the XML/DBC [7] API. We propose the design of the XConstraint Checker module.

¹ By global XConstraints we mean global semantic integrity constraints affecting multiple XML databases.

² Sub XConstraint is a XML constraint, expressed as an XQuery, local to a single site.

XML/DBC [7] is the standard XML XQuery API that facilitates access to XML based data products. In our case, XML/DBC API executes the sub XConstraints corresponding to the remote sites. The XConstraint Checker gathers the results obtained from the sub XConstraints and makes a decision if a constraint is violated. Only in the event of no constraint being violated, the XUpdate statement is executed.

By the very nature of this paper, describing XConstraint Checker architecture, algorithms, proofs, examples, and implementation details, we will not be

able to provide in-depth description of all the aspects because of space limitations. However, interested readers are encouraged to refer to the extended version of this paper[10]. The rest of the paper is organized as follows: In Section 2, we give example XML databases that will be referred throughout the paper. We also give the syntax of XUpdate language and introduce our notations for defining global XConstraints. In Section 3, we give internal architecture of the XConstraint Checker and explain in detail, steps involved in our constraint checking procedure with examples. Finally, we compare our work with other peer's work and offer our conclusions in Section 4.

2. PRELIMINARIES

Here we give an example healthcare XML database and explain the notations of XUpdate. We also introduce our notation for defining XConstraints.

2.1 Example Database

Consider a sample *healthdb.xml* represented in the tree form in Figure 2. It gives the logical representation of the HEALTHDB XML databases. Physically, information is distributed across multiple sites

2.2 XUpdate

XUpdate is the language extension to XQuery to accommodate insert, replace, delete and rename operations. In

[11], the XUpdate language syntax and semantics are presented. Below, we show a sample XUpdate occurring on the XML tree (node 11) of Figure 2

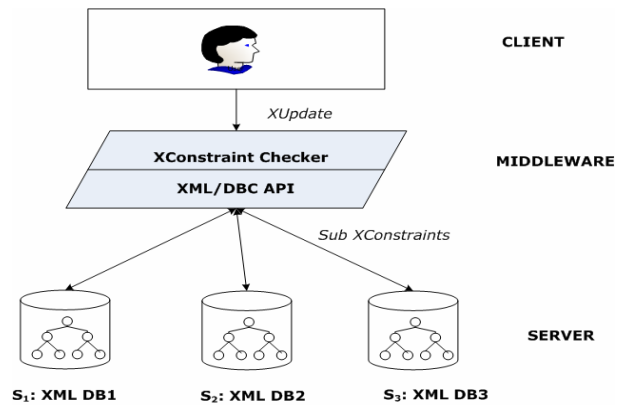


Figure 1: Overview of system

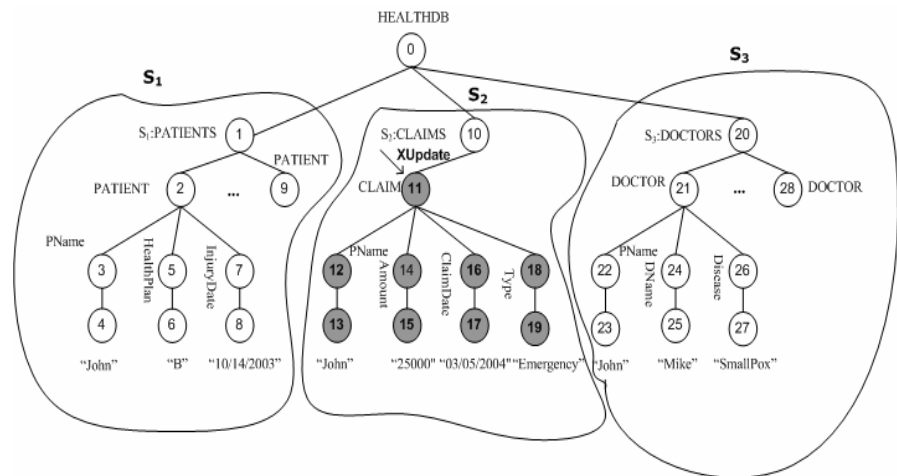


Figure 2: Tree representation of healthdb.xml

```

XU1 =
FOR $claim in document
("healthdb.xml")/HEALTHDB/S2:CLAIMS
UPDATE $claim
{
INSERT <CLAIM>
    <PName>John</PName>
    <Amount>25000</Amount>
    <ClaimDate>03/05/2004</ClaimDate>
    <Type>emergency</Type>
</CLAIM>
}

```

A detailed description of the XUpdate language can be found in [10], [11]

2.3 XML Constraint Representation

Throughout the paper, we denote semantic integrity constraints for XML database as *XConstraints*. *Global XConstraints* are the constraints spanning multiple XML databases. Here we give the constraint representation for global XConstraints.

A datalog rule (expressed as Head \leftarrow Body) without a Head clause is referred to as a denial. It is customary to represent integrity constraints in the logic databases as range restricted (safe or allowed) denials.

Definition 2.1: In order to represent global XConstraint in the context of XML database as query evaluation, we consider global XConstraint in the form of range restricted denials (datalog style notation) given as $C \leftarrow X_1 \wedge X_2 \wedge \dots \wedge X_n$, where C is the name of the global XConstraint and each X_i is either an XML literal or Arithmetic literal. ■

We define both XML literal and arithmetic literal below. The definition of XML literal is chiefly inspired from [2], [5]. Semantics for representing key constraints for a single XML database are given in [2], [5]. We extend their semantics by introducing user defined variables, term paths and XML literals for representing global XConstraints for multiple XML databases.

Definition 2.2: *XML literal* is defined as follows:

$$X_i : (Q_i, (Q_i', [V_{i1} = t_{i1}, V_{i2} = t_{i2}, \dots, V_{ik_i} = t_{ik_i}]))$$

Using the syntax of [2], [5], Q_i , Q_i' and $t_{i1}, t_{i2}, \dots, t_{ik_i}$ are path expressions corresponding to X_i . $V_{i1}, V_{i2}, \dots, V_{ik_i}$ are user defined variables corresponding to $t_{i1}, t_{i2}, \dots, t_{ik_i}$. Q_i is called the context path, Q_i' the target path and $t_{i1}, t_{i2}, \dots, t_{ik_i}$ are the term paths. Q_i identifies the set of context nodes, c and for each c , $V_{i1}, V_{i2}, \dots, V_{ik_i}$ are the set of user defined variables corresponding to the term paths, $t_{i1}, t_{i2}, \dots, t_{ik_i}$, reachable from c via Q_i' . ■

Definition 2.3: *Arithmetic literal* is defined as: $x \theta y$, where x is one of the variables occurring in XML literals; θ – is a comparison operator ($=, <, >, <=, >=, <>$); y is either a variable occurring in XML literal or a *constant*. Joins between nodes are expressed either as equality between two variables in an arithmetic literal or by having the same variable name appear in different XML literals with in the same global XConstraint. Variables with the same name cannot appear in the same XML literal. ■

Definition 2.4: A XML tree T is said to satisfy a global integrity constraint (global XConstraint), C , if and only if the conjunction of X_1, X_2, \dots, X_n evaluates to false. ■

Note that each Q_i, Q_i' , user defined variables and the term paths corresponding to each XML literal, X_i , have the site information (referred as S_j) and can only refer to a single site. However, a global XConstraint has one or more XML literals and hence can refer to multiple XML databases. In case of Arithmetic literal, $x \theta y$, the two variables x and y could belong to different sites. If two variables are not the leaf nodes, the equality join among the two variables is similar to the node equality considered in [2].

Example 2.1: Consider two global XConstraints C_1 and C_2 defined on *healthdb.xml*. Constraint C_1 states

that a patient with HealthPlan ‘B’ diagnosed with ‘SmallPox’ may not claim more than 40000 dollars. Constraint C_2 states that a patient with HealthPlan ‘B’ may not file a claim of type ‘Emergency’.

```
C1:-
  (//S1:PATIENTS,
   (./PATIENT, [name=./PName,healthplan=./HealthPlan])),
  (//S2:CLAIMS, (./CLAIM, [name=./PName,amount=./Amount])),
  (//S3:DOCTORS, (./DOCTOR, [name=./PName,disease=./disease])),
  healthplan = 'B',disease = 'SmallPox',amount > 40000.
```

```
C2:-
  (//S1:PATIENTS, (./PATIENT, [name=./PName,healthplan=./healthplan])),
  (//S2:CLAIMS, (./CLAIM, [name=./PName,type=./type])),
  healthplan = 'B',type = 'Emergency'.
```

For the Figure 2, C_1 is satisfied, C_2 is not. C_1 is satisfied for the *healthdb.xml* as one of the arithmetic literals *amount* (node 14, value = 25000) > 40000 returns false and hence the whole conjunction for C_1 evaluates to false. C_2 is not satisfied as the conjunction for C_2 evaluates to true. Arithmetic literal, *healthplan* (node 5, value = 'B') = 'B' evaluates to true and similarly, *type* (node 18, value='Emergency') = 'Emergency' evaluates to true and hence the whole conjunction for C_2 evaluates to true.

3. XCONSTRAINT CHECKER

First, we lay down the assumptions of the system and then present the detailed architecture of the XConstraint Checker.

3.1 Assumptions

The assumptions we make for the XConstraint Checker are:

1. We consider a restricted set of XUpdate language with out losing the generality of the approach. We permit the following SubOPs[10]: DELETE \$child, INSERT content [BEFORE | AFTER \$child] and REPLACE \$child with \$content.
2. We restrict the updates to *elementary updates*. The elementary update considers: (i) updates occurring only on one single node of an XML tree (ii) updates with only one SubOP at a time. However, note that any update can be equivalently transformed to a set of elementary updates; therefore, we do not lose the generality of the approach.

3.2 XConstraint Checker Architecture

XConstraint Checker interacts with the rest of the system as shown earlier in Figure 1. The internal architecture of the XConstraint Checker is presented in Figure 3. The XConstraint Checker consists of the following modules: XUpdate Parser, XMetadatabase (repository of global XConstraints), XMeta Extractor, and XConstraint Decomposer. The overall process of constraint checking is explained in the following four steps.

STEP 1: The user issues a XUpdate statement on one of the sites. Say, a user issues a XUpdate statement, XU_1 (given in Section 2.2) on site S_2 . Figure 2 gives the modified tree representation of the *healthdb.xml*, if the update is successful. The nodes affected by the XUpdate are shown in filled circles.

STEP 2 (XUpdate Parser): The XUpdate Parser parses the given XUpdate statement and identifies the XML node being modified. The output from this step is the XML Node Value List (XNVL). $XNVL = N(a_1=v_1, a_2=v_2, \dots, a_n=v_n)$, where N is the node being updated and is obtained from the $\$binding$ in the XUpdate syntax, v_1, v_2, \dots, v_n are the values being updated corresponding to the attributes a_1, a_2, \dots, a_n . a_1, a_2, \dots, a_n are either the XML sub elements or XML attributes being updated and are obtained from the content of the XUpdate statement (Section 2.2). For the running example,

```
XNVL = {/HEALTHDB/S2:CLAIMS/CLAIM( PName = 'John', Amount = 25000,
    ClaimDate = '03/05/2004', Type='Emergency' ) }
```

STEP 3 (XMeta Extractor): Let $XU \downarrow$ denotes the path involved in executing the XUpdate statement, XU on the XML tree T . Similarly, $C \downarrow$ denotes path in defining the constraint C . We say that a XUpdate, XU might violate a constraint C if, $XU \downarrow \cap C \downarrow$ is not empty. Note C_1 and C_2 are given in Example 2.1. $XU \downarrow$ corresponds to the following nodes: $\{11,12,13,14,15,16,17,18,19\}$, $C_1 \downarrow$ matches $\{3,4,5,6,12,13,14,15,22,23,26,27\}$ and $C_2 \downarrow$ matches $\{3,4,5,6,12,13,18,19\}$ (refer Figure 2). $XU \downarrow \cap C_1 \downarrow$ is not empty and $XU \downarrow \cap C_2$ is also not empty; hence, both the constraints might violate update statement. If a global schema or a global DTD is given, we can identify the list of global XConstraints that might be violated by simply consulting the global DTD.

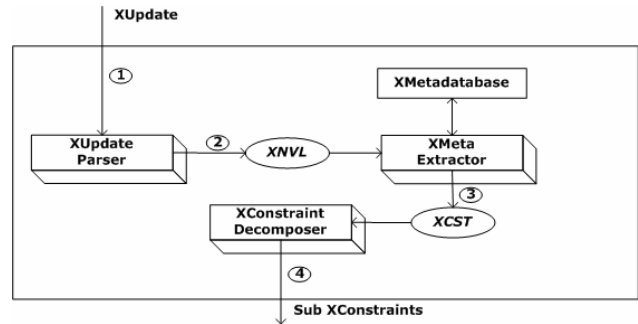


Figure 3: XConstraint checker internal architecture

The *XMeta Extractor* identifies the list of constraints being affected by the XUpdate and constructs the XConstraint Source Table (XCST). $XCST(C_i) = \langle C_i, list(S_j) \rangle$, where C_i is the constraint identifier and $list(S_j)$ is the list of sites being affected by C_i . The XMeta Extractor sends the XCST to the XConstraint Decomposer.

STEP 4 (XConstraint Decomposer): The XConstraint Decomposer decomposes a global constraint C_i in to a set of sub XConstraints, C_{ij} on the basis of locality of sites. C_{ij} is the sub XConstraint corresponding to constraint, C_i and site S_j . We present algorithmic description of generating C_{ij} s, proof of its correctness, examples and implementation details in the extended version of this paper [10]. For the running example, C_{11} , C_{12} , C_{13} , C_{21} and C_{22} are generated.

```

C11 = for $var1 in document("healthdb.xml")//S1_PATIENTS/PATIENT
      where $var1/PName = "John" and $var1/HealthPlan = "B"
      return 1
C12 = return 1 if {"John" = "John" and 25000 > 40000} else return 0
C13 = for $var2 in document("healthdb.xml")//S3_DOCTORS/DOCTOR
      where $var2/PName = "John" and $var2/Disease = "SmallPox"
      return 1
  
```

So, $C_1 = C_{11} \wedge C_{12} \wedge C_{13}$. In this example, $C_{11} = 1(\text{true})$, $C_{12} = 0(\text{false})$ and $C_{13} = 1(\text{true})$. The conjunction of C_{11} , C_{12} and C_{13} evaluates to *false*. Hence the update statement does not violate constraint C_1 (from Definition 2.4). Similarly, C_{21} and C_{22} are also generated.

4. RELATED WORK AND CONCLUSIONS

Related Work: The idea of keys and foreign keys for XML was introduced in [2], [5]. The basic approach is to express constraints using path expressions. We have extended the approach of [2], [5] with

datalog style notations and also used the concepts from [8] in representing XConstraints. The area of constraint checking for XML is relatively new and very few research results exist in this area. To our knowledge we have not seen any research on semantic integrity constraint checking for multiple XML databases. Research on validating keys for XML can be found in [1], [3] and [6]. To our knowledge, the only work closest to ours is SAXE [9]. SAXE executes only those XUpdates that would preserve the consistency of the XML document with respect to a particular schema. The underlying idea of SAXE is to generate constraint check sub queries. The constraint check sub queries check if the given XUpdate statement violates the consistency of the XML document. The XUpdate statement in SAXE is executed only if it is safe. Hence SAXE avoids any potential rollbacks. We also take a similar route. However, SAXE does not consider semantic integrity constraint checking for multiple XML databases.

Conclusions: We have presented the architecture of XConstraint Checker. XConstraint Checker is part of a middleware module, which determines if a XUpdate statement violates any global XConstraints. In a nutshell, we have: (i) introduced a notation for representing XConstraints, and (ii) proposed architecture for XConstraint Checker, which given an XUpdate statement, list of global XConstraints, generates at compile time, sub XConstraints to be executed locally on remote sites. XConstraint architecture is efficient for two reasons: a) checks for constraint violations without actually updating the database and hence avoids any potential rollbacks, and b) achieves speed as the sub XConstraints are executed in parallel.

We plan to extend our constraint checking mechanism to global XConstraints involving *aggregate literals* (sum, max, min, avg and count). Also, note that for each global XConstraint that could be violated, multiple sub XConstraints are generated. Hence, we have a large number of sub XConstraints when we consider all the set of global XConstraints. Therefore, efficient ordering of sub XConstraints for executing on remote sites would optimize the constraint checking mechanism. To achieve this, we plan to introduce *XConstraint Optimizer* module in the XConstraint Checker.

5. REFERENCES

- [1] M.Benedikt, C.Y. Chan,W. Fan,J.Freire and R.Rastogi.Capturing both Types and Constraints in Data Integration. ACM SIGMOD, 2003.
- [2] P. Buneman, S. Davidson, W.Fan, C.Hara, and W.Tan. Keys for XML. In WWW10,2001, pp.201-210.
- [3] B. Bouchou, M. Halfeld-Ferrari-Alves, and M. Musicante.Tree Automata to Verify XML Key Constraints. WebDb 2003.
- [4] Yi Chen, Susan B.Davidson, Carmem S. Hara, Yifeng Zheng, RRF: Redundancy Reducing XML Storage in Relations. VLDB 2003.
- [5] Y.Chen, S.B. Davidson, and Y.Zheng. Constraint Preserving XML Storage in Relations. In WebDB, 2002.
- [6] Y.Chen,S. Davidson,Y. Zheng. XKvalidator:A Constraint Validator For XML. Proceedings of ACM CIKM, 2002.
- [7] G. Gardarin, A. Mensch, T. Tuyet Dang-Ngoc, L. Smit. Integrating Heterogeneous Data Sources with XML and XQuery.Proceedings of the 13th International Workshop on Database and Expert Systems Applications, 2002.
- [8] A.Gupta and J. Widom.Local Verification of Global Integrity Constraints in Distributed Databases.Proceedings of the ACM SIGMOD 1993.
- [9] B. Kane, H. Su, and E.A. Rundensteiner, Consistently Updating XML Documents using Incremental Constraint Check Queries. Workshop on Web Information and Data Management (WIDM'02), Nov. 2002. page 1-8,2002.
- [10] P. Madiraju, R.Sunderraman, and S.B. Navathe. Semantic Integrity Constraint Checking for Multiple XML databases. Unpublished manuscript available from http://tinman.cs.gsu.edu/~cscpnmx/research/XML/XConstraint_TechReport.pdf
- [11] I. Tatarinov, Z. G. Ives, A. Y. Halevy, S.Daniel. Updating XML. ACM SIGMOD 2001.