

# Access Control for XML - A Dynamic Query Rewriting Approach

Sriram Mohan  
Computer Science Dept.  
Indiana University  
srmohan@cs.indiana.edu

Arijit Sengupta  
Raj Soin College of Business  
Wright State University  
arijit.sengupta@wright.edu

Yuqing Wu  
School of Informatics  
Indiana University  
yuqwu@indiana.edu

## ABSTRACT

Being able to express and enforce role-based access control on XML data is a critical component of XML data management. However, given the semi-structured nature of XML, this is non-trivial, as access control can be applied on the values of nodes as well as on the structural relationship between nodes. In this context, we adopt and extend a graph editing language for specifying role-based access constraints in the form of security views. A Security Annotated Schema (SAS) is proposed as the internal representation for the security views and can be automatically constructed from the original schema and the security view specification. To enforce the access constraints on user queries, we propose Secure Query Rewrite (SQR) - a set of rules that can be used to rewrite a user XPath query on the security view into an equivalent XQuery expression against the original data, with the guarantee that the users only see information in the view but not any data that was blocked. Experimental evaluation demonstrates the efficiency and the expressiveness of our approach.

## Categories and Subject Descriptors

H.2.7 [Database Administration]: Security

## General Terms

Management, Security, Performance

## Keywords

XML, Access Control, Security View, Query Rewrite

## 1. INTRODUCTION

XML is one of the most extensively used data representation and data exchange formats. Much of the research on XML has focused on developing efficient mechanisms to store, query and manage XML data either as a part of a relational database or using native XML stores. However, hiding sensitive data is as important as making the data efficiently available, as has been emphasized and studied for decades in relational databases.

### 1.1 Motivating Example

Consider the problem of developing and conducting tests through a Course Management System such as Oncourse<sup>1</sup>.

<sup>1</sup>Oncourse is developed as part of the Sakai project involving

Online tests and quizzes are stored in XML in Oncourse using the IMS-QTI schema<sup>2</sup>. A highly simplified version of this structure is shown in Figure 1.

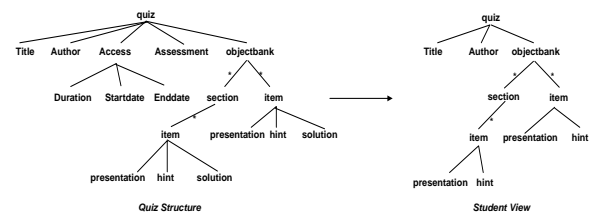


Figure 1: Simplified Tree Structure for an Online Quiz and the Security View

EXAMPLE 1.1. Several access constraints can be envisioned in the online quiz application. An author (instructor) should have access to all elements of quizzes he/she writes (full access). A student should only have access to current quizzes in courses that he/she is registered for, but not have access to the solutions (conditional and unconditional removal). Moreover, instructors other than the authors should have access to the questions as well as to the solutions, but potentially without the course-specific structuring (conditional and unconditional restructuring). Finally, for summarizing purposes, a statistician may have access to all the solutions. But even for someone who may have permission as both a student and as a statistician, he/she should not be able to figure out the solution to each individual question (removal of association). If materialized views are used to implement the above role-based security policy for the student view, the view generated would be as shown in Figure 1(b).

All the security constraints in Example 1.1 are not uncommon, yet, none of the existing techniques support them without actually generating or materializing the “views”. However, given that the data may constantly change, and that usually there are multiple security levels in a system with each level containing multiple security views, even incrementally maintaining the security views is sometimes non-realistic in a real-time system. An alternative is to maintain only virtual security views and enforce the access constraints via query rewrites, as proposed in [1]. However, the security view specification language needs to be enriched for DBAs to be able to define views similar to those in Example 1.1.

University of Michigan, Indiana, MIT, Stanford, OKI and the uPortal Consortium.

<sup>2</sup>IMS Global Learning Consortium - <http://www.msglobal.org>

## 1.2 System Infrastructure

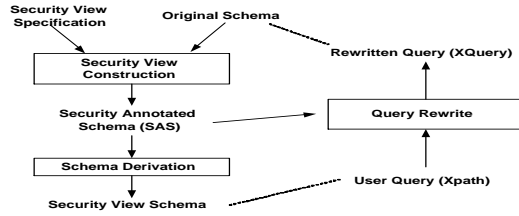


Figure 2: The Infrastructure of the Security View Based Query Answering System

The infrastructure of our security-view based query answering system is as shown in Figure 2. Taking the original XML schema and security view specification sequence in SSX as input, the **Security View Construction** component (on the left) constructs a Security Annotated Schema (SAS). SAS is an internal representation in our system. The security view schema can be trivially obtained from an SAS and it is the only schema that is made available to the corresponding user group. In our system the security views are not materialized by default. The **Secure Query Rewrite** component (on the right) rewrites the user queries (in XPath) into equivalent target queries (in XQuery) against the original schema, using the information in the SAS. These rewritten queries are then evaluated against the base data.

## 2. SECURITY SPECIFICATION AND ENFORCEMENT

Given security constraints such as those in Example 1.1 and other similar cases, we introduce our Security Specification Language for XML (SSX) in the form of a set of graph editing primitives.<sup>3</sup> In SSX each primitive takes an XML schema tree as input, and outputs an XML schema tree. The parameters and functions of the primitives are defined as follows (parameters within square brackets are optional):

- create(destSPE, newName)** creates a new element with tag ‘newName’, as a child of each element that matches the ‘destSPE’ in the input schema.
- delete(destXPath)** removes the sub-trees rooted at the elements that matches the ‘destXPath’ in the input schema.
- copy(sourceXPath, destSPE, [newName], [scope], [preserve])**: For each element that matches the ‘scope’, the **copy** primitive creates an identical copy of the sub-trees rooted at the nodes that match the ‘sourceXPath’ in the original schema with respect to the ‘scope’, and makes them the children of the elements that match the ‘destSPE’ in the input schema. If a new name is provided, a new element tag is assigned to the root element of the copied sub-trees. The default value for ‘scope’ is ‘/’. ‘Preserve’ specifies if the copy primitive should preserve (the default) or break the document order between instances being copied.
- rename(destSPE, newName)** assigns a new tag to the elements that matches the ‘destSPE’ in the input schema.

A security view specification is then written in the form of a sequence of these primitives. Each primitive takes the result of the subsequence in front of it as input. The final

<sup>3</sup>SPE - A Simple Path Expression is defined as an XPath expression without branching predicates.

result is the security annotated schema (SAS) for the SSX sequence.

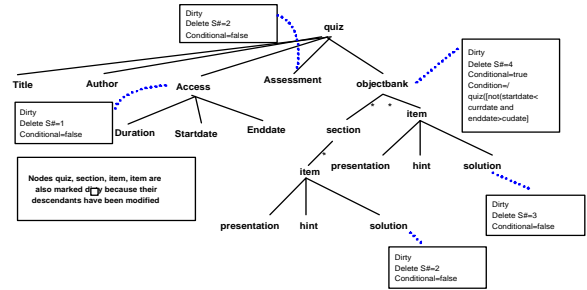


Figure 3: The Security Annotated Schema for the Student Security View shown in Fig 1

To facilitate query answering and rewriting, we propose an internal representation - Security Annotated Schema (SAS) in the form of annotations to represent the schema transformation specified by a SSX sequence. Annotations are associated with the element node that was modified and reflects the actual changes performed on the original schema tree structure. SAS introduces the following annotations to reflect the SSX primitives that specify the access constraints:

- Delete
- Newnode
- Scope Stamp and Dirty Stamp
- Chronological Operation Sequence#

We introduce algorithms to automatically construct the SAS from the SSX primitives. The security annotated schema for the student view is shown in Figure 3 and the security view for the same can be trivially derived from the SAS.

Rather than creating and maintaining materialized security views, we choose to rewrite the user queries (in XPath) to queries (in XQuery) that reflect the security constraints and are evaluated against the source data. The Secure Query Rewrite process (SQR) is rule-based, and is defined via a recursive rewrite function that translates an user XPath query on the security view to an XQuery expression against the original data.

Experimental evaluations reveal that our approach is effective and efficient. A complete discussion of SSX, SAS and query rewrites(SQR) and the algorithms and rewrite rules are available at [2].

## 3. CONCLUSION

We introduce a framework for specifying and representing complex security constraints for XML and enforcing the security constraints during query evaluation, via the query rewrite process. This framework is the first to introduce the capability to specify and enforce complicated security policies on both sub-trees and structural relationships in an XML document.

## 4. REFERENCES

- [1] W. Fan, C.-Y. Chan, and M. Garofalakis. Secure XML querying with security views. In *SIGMOD*, 2004.
- [2] S.Mohan, A.Sengupta, Y.Wu, and J.Klingsmith. XML access control, at <http://www.cs.indiana.edu/~access>.