A MODEL FOR CONTROLLING ACCESS TO XML DOCUMENTS

Kh. Azizur Rahman[1], Md. Rafiqul Islam[1], Mohd. Noor Md. Sap[2], AYM Mostafa[1] & Hari Prasad Baral[1]

[1]Computer Science & Engineering Discipling, Khulna University, Khulna, Bangladesh.
[2]Faculty of Computer Science & Information System,
University Technology of Malaysia (UTM), Johor, Bahru, Johor, Malaysia.

Abstract:
There are often cases where the data must be protected from possible threats as the data may contain confidential information. Our goal is to combine practical data representation of XML with various security features. We have proposed XAR (XML Access control model using RDB) an access control model for controlling access to XML documents using RDB (relational database). In this model, XML documents are stored in RDB. Through XAR user can access only those portions of an XML document to which he or she is given permission. We enhance XML with a sophisticated access control mechanism that enables the client not only to securely browse XML documents but also to securely update each document element. We also have provided a graphical front end for the administrator or owner, which will allow specify and manage access rules and policies to XML documents or elements. Our study of performance of XAR shows that the response time and memory utilization of XAR is also very efficient. It is also able to handle big sized XML Documents where most of the traditional systems are disable.

Keywords: XML documents, access control, access rule.

1.0 Introduction

XML (eXtensible Markup Language) is the latest promising technology that has powerful applications used in the Intra or Internet world, particularly for information interchange and the management, display and organization of data. The main benefit of XML over HTML is related to the possibility of defining tags, nested document structures, and DTDs (Document Type Definitions) or Schema describing the structure of a set of documents. XML documents can contain information at different degrees of sensitivity. In some cases, same access control policy may be applied to a set of documents. In other cases, different access control policies may be applied to different portions of the same document. So the simple method of protecting an entire document at file level is not satisfactory for XML documents. XML documents need specific level (fine-grained) access control system, which will enforce the access control rules to the XML element level. We have proposed an access control system for XML using RDB database, so that the access of XML document be more efficient of memory and time than traditional access control system. This method extracts information from an XML schema so that documents based on the schema can be mapped into multiple RDB tables. Under this system, the access rule of individual XML elements can be assigned to different users, and users can get privileges according to the access rule defined for them. This access control scheme is much desirable in situation where XML documents will be updated by different users of a distributed environment. A front side Graphical User Interface for the administrator or owner has also been developed to define the access rules and access policies to the XML documents or elements.

The remaining of this paper is organized as follows. In Section 2.0 basic concepts of XML have been briefly described. Protection system and protection requirement for XML documents are also described in this

section. In Section 3.0 we introduce the overall design XAR. We have shown a performance study in section 4.0. At last we conclude in Section 4.0.

2.0 XML Basics

XML evolved from standard generalized markup language (SGML) and became a World Wide Web Consortium (W3C) recommendation in February 1998 [1]. One of XML's main goals is to separate the structure and content of a document from its presentation. An XML document is constructed with elements. Elements can be nested at any depth and can contain other elements (sub elements). An element contains a portion of the document delimited by two tags: the start tag, at the beginning of the element, of the form <tag-name>, and the end tag, at the end of the element, of the form </tag-name>. Empty elements, such as <tag-name/>, are also possible. An example of XML document containing information on a company department is shown in figure 1. A list of attributes can also be specified for an element.

```
<department id="production">
<employee id="A101">
  ...
</employee>
  ...
<employee id="A123" manager="A101">
<name>
<fname> Nil </fname> <lname>Armstrong  </lname>
</name>
<address>
<street> 32,moon street </street>
 <email mailto="nil@yahoo.com"/>
</address>
<salary > 8000 </salary>
</employee>
...
<employee id="A150" manager="A123">
...
</employee>
</department>
```

Figure 1. An example of XML Document

A document type definition (DTD) can be attached to XML documents, specifying the rules that XML documents may follow. A DTD is composed of two parts: *element declarations* and *attribute list declarations*. DTDs were the original means of specifying the structure of an XML document and a holdover from XML's roots as a subset of the Standarized and Gereral Markup Language (SGML). DTDs have a different syntax from XML and are used to specify the order and occurrence of elements in an XML document.

XML Schema is a newer approach to validate a XML document that contains a mechanism for constraining the allowable content of a class of XML document. The W3C XML schema recommendation provides a sophisticated means of describing the structure and constraints on the content model of XML documents. The XML schema recommendation's was released in May 2001 [4]. It would allow authors to place

restrictions on the allowable element content and attribute values in terms of primitive data types found in languages such as SQL and Java.

XPath is a language for addressing parts of an XML document that utilizes a syntax that resembles hierarchical paths used to address parts of a file system or URL. XPath also supports the use of functions for interacting with the selected data from the document. It provides functions for the accessing information about document nodes as well as for the manipulation of strings, numbers and Booleans. XPath is extensible with regards to functions that allow developers to add functions that manipulate the data retrieved by an XPath query to the library of functions available by default. XPath uses a compact, non-XML syntax in order to facilitate the use of XPath within URLs and XML attribute values. XPath is designed to operate on a single XML document which it views as a tree of nodes and the values returned by an XPath query are considered conceptually to be nodes. The types of nodes that exist in the XPath data model of a document are text nodes, element nodes, attribute nodes, root nodes, namespace nodes, processing instruction nodes, and comment nodes.

2.1 Protection of XML Documents

In this section, requirements of protection of XML documents and some proposed protection system to XML documents will be discussed.

2.1.1 Protection Requirements

Protection of XML documents in an XML source demand new requirements with respect to data protection in conventional databases. In the following sections, these are discussed briefly.

Granularity level authorization: An XML document is structured into elements and attributes. Elements can contain elements (sub elements) in turn, resulting in a hierarchical structure. Moreover, XML documents are inter-linked since their elements can refer to other related elements in the document through IDREF(S) (reference to identify one or more specific element which have ID attribute) attributes. Very often, different (sub) elements, attribute(s), or link(s) within the same document have varying protection requirements. In example shown in figure 1, information about employee in a department can be made available to everyone, whereas information about who is the manager of whom (represented as a link between employee elements) could be kept hidden from most subjects and made visible only to a restricted number of authorized subjects. As a consequence, authorizations for XML documents should be associated with protection objects at different granularity levels. In particular, the following protection object granularities are identified:

- Document or DTD or Schema
- Group of documents
- (Sub) element

Protection of valid and well-formed documents: The presence of DTDs or Schemas and of valid and well-formed documents in a source suggests the specification of authorizations at two different levels of abstraction:

- DTD (schema) level: a DTD defines the structure of a set of XML documents and can be considered as a schema for them. In analogy with authorization models for conventional databases, authorizations (at different granularity levels) can thus be specified at the DTD level and propagated to all documents that are valid instances of the DTD, according to the DTD-to-instance relationship.
- Document (instance) level: authorizations (at different granularity levels) are defined at the document level and propagate on the considered document only, according to the element-to-sub element and the element-to-attribute/link relationships.

2.1.2 Protecting XML Documents - A New Approach

Conventional HTML Tagging is aimed at defining page rendering. And authorization cannot be applied at granularity level to HTML documents. For example, the Apache server (www.apache.org) allows the specification of access control lists via a configuration file (access.conf) containing the list of users , hosts (IP addresses), or host or user pairs, which must be allowed or forbidden connection to the server. Users are identified by user-id and group-names and passwords, to be specified via Unix-style password files. By specifying a different configuration file for each directory on the Web server's disk, it is possible to define authorizations on a directory basis; files belonging to the same directory are subject to the same authorizations. The specification of authorizations at the level of single file (i.e., Web pages) is quite awkward, while it is not possible to specify authorizations on portions of files. This limitation forces protection requirements to affect data organization at the file system level. But the semantics of XML is such that we can authorize a user to an XML document to the granularity level. In [2] an access control model has been presented. However this model is based on DOM (Document Object Model) and it is not suitable for large XML documents. In [6] application programming interface is used to propose an access control model. Though the proposed model provides efficient storage, it is not suitable for response time. We implement the access control model using relational database, so that it can be handled with less memory and provide better response time. Our proposed model adopts the access control rule format presented in [7], which specifies the access control rules in a 5-tuple format {subject, object (XPath), mode, Grant or deny, grantor}. The 5-tuple-format rule specifies a subject is granted/denied permission to perform the action (read or write) on the XML object. By specifying the XML element object in XPath, the access control system is able to enforce the access control rules to the tag or element level. By using the ownership access control scheme, it provides much flexibility in access control compared to the traditional access control system.

3.0 Overall Design of XAR

The overall design of XAR is shown in figure 2. The operation is explained below.

3.1 System Architecture of XAR

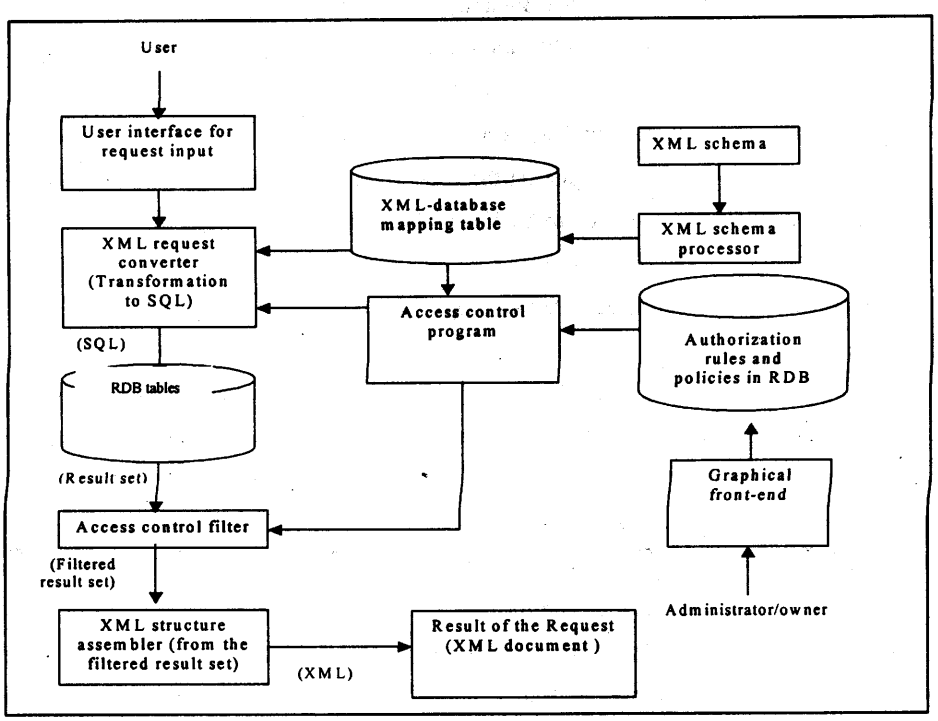

Fig 2. Overall design of the proposed access control model

i) At first from The XML schema an XML-Database mapping table is generated. This mapping table is used for transformation between XML documents and the relational database. From an XML file XML data are extracted and stored into the relational database tables according to the mapping information table that is generated earlier. The mapping procedure is described in section 4.1.

ii) Access rules are created by administrator or by the owners of the XML elements. In our model, both positive and negative authorization can be specified. Positive authorizations indicate permissions whereas negative authorization indicates denial for access.

Authorizations are defined as follows.

DEFINITION 4.1 (Authorization): An authorization is a 5-tuple {sub, obj, mode, sign, grantor} where:

sub ∈ U is the user to whom the authorization is granted;

obj ∈ O is the object(XPATH) to which the authorization refers;

mode ∈ M is the access mode(read/write)for which the authorization is granted;

sign ∈ {+,-} indicates whether the authorization is positive (+) or negative (-);

grantor ∈ U is the user who granted the authorization;

Tuple {s,o,m,+,g} states that user s has been granted access mode m on object o by user g. Tuple {s,o,m,-,g} states that user s has been forbidden access mode m on object o by user g.

iii) User can request to view or update an XML document or portion of the document. Authorization requests are defined as follows.

DEFINITION 4.2 (Request): A request is a 3-tuple {sub, obj, mode} where:

sub ∈ U is the user who requests ;

obj ∈ O is the object(XPATH) to which the user has requested;

mode ∈ M is the access mode(read/write)for which the user has requested;

Tuple {s,o,read} states that user s has requested to perform read operation on object o. Whereas Tuple {s,o,write} states that user s has requested to perform write operation on object o.

iv) In response to the user requests for an XML document, the XML request processor will translate the request into SQL for the relational database based on the mapping information. At the same time, the access controller will also be activated to apply access rules to the relevant XML elements. Access control is first applied in the XML request processor. For example if a user cannot view all instances of an XML element, then the request processor will not generate the SQL for that element. If the access rules are complicated, the access controller will generate an access filter. The access filter is applied to the result set returned by the relational database, and unauthorized elements are filtered out before further processing. Finally, the filtered data are assembled back into XML documents through an XML structure assembler and the final XML document is returned to the user.

In the following sections Transforming procedure of an XML document to relational database, ownership access control rule, request processing and Graphical front-end interface will be briefly discussed.

3.2 Transformation of XML documents to Relational Database

The first issue in mapping between XML and a set of relations is the correspondence between a relation and some portion of the XML document's hierarchy. Once a mapping to tables has been established, a means of converting from tabular output to XML must be determined. In order to transfer data from XML documents to a relational database, it is necessary to map the XML document schema (DTD/ XML Schema) to the database schema. The data transfer software is then built on top of this mapping. The software may use an XML query language (such as XPath, or XQuery) or simply transfer data according to the mapping

information. Before transferring data to the database, the document is first transformed to the structure expected by the mapping; the data is then transferred.

3.2.1 Mapping XML Schemas to Database Schemas

Mappings between XML schemas and database schemas are performed on element types, attributes, and text. We used the table-based mapping to transfer data between an XML document and a relational database. It models XML documents as a single table or set of tables. The table-based mapping is useful for serializing relational data, such as when transferring data between two relational databases. In this model, element types with attributes, element content, or mixed content are called "complex element types". Element types with PCDATA-only content are "simple element types".
To generate a relational schema from a DTD we will follow the following algorithm.

Algorithm 3.1 generating a relational schema from a DTD or XML schema

1. For each complex element type, create a table and a primary key column.
2. For each element type with mixed content, create a separate table in which to store the PCDATA, linked to the parent table through the parent table's primary key.
3. For each single-valued attribute of that element type, and for each singly occurring simple child element, create a column in that table. If the child element type or attribute is optional, make the column null able.
4. For each multi-valued attribute and for each multiply occurring simple child element, create a separate table to store values, linked to the parent table through the parent table's primary key.
5. For each complex child element, link the parent element type's table to the child element type's table with the parent table's primary key.

3.3 Ownership Rule of Access Control

By specifying the XML element object in XPath, the access control system is able to enforce the access control rules to the XML element level. However like most traditional access control model, it relies on the administrator to set the access control rules for all the XML elements. Based on the traditional access control system, we would need the administrator to constantly update the access control rules in order to keep the latest XML document secure. In the proposed model, the administrator can assign ownership of individual XML elements to different users. Users then can set the access rules for the elements they own.
Owner's rules are only effective in the owner's scope. Owner's scope refers to the XML elements that the user owns. Figure 3 illustrates the ownership system.

```
<Root>
 <User ID="1">
    <Item> Item1</ Item>
</ User >
< User ID ="2">
    < Item> Item2</ Item>
</ User >
</Root>
```

a) Sample XML document

```
< User ID="1">
    < Item > Item 1</ Item >
</ User >
```

b) Elements owned by user 1

```
< User ID="2">
    < Item > Item 2</ Item >
</ User >
```

c) Elements owned by user 2

Figure 3: Sample XML documents and owners elements

The ownership access control scheme can provide greater flexibility because user can set the access control rules for the elements they own. So when he or she updates the XML document, he or she can update the access control rules at the same time. Furthermore these rules are only effective in owner's scope, so they will not temper the existing rules for other elements.

Under the ownership access control scheme, a special column called owner column is added to the database for each XML element to record the owner of the element. When system applies owner's access rules, it adds an additional condition to check whether the XML element belongs to the owner.

Under the ownership access control scheme, when a user creates a new XML element or upload a new XML file, he or she will automatically become the owner of the elements or documents. User can also assign the ownership of his or her own elements to other users.
In additional to the owner's rule, two more rule types will be implemented. They are "Low priority" and "High priority" rules. The administrator will set these rules. "Low priority" rule will be used as default rules for XML elements if no other access control rules have been specified for them. "High priority" rules will have the highest priority; they will be set by the administrator to enforce certain strict rules.

If conflict exists for different rules with the same priority, it is resolved by the most specific rule as presented in [2]. That means, if the object specified by one rule is more specific than the other, this rule will have higher priority.

3.4 Request Processing

When a user submits a request for an XML document or portion of the document, the access controller will activate to check the access rights of the user and apply access rules accordingly. The access controller will first extract the user or group information about the user. The user or group information records the identity of the user and the group he/she belongs to. For example, if user "user1" belongs to group "users", which in

turn belongs to group "allusers", then the user or group information should be recorded as follows: user1->users->allusers. The user or group information is automatically generated when user login to the system.
Next, the access controller will find out what are the object elements the user requests. This information is extracted from an XML request processor, which will parse the XML query and generate the relevant information.

In order to find out the relevant owner type access rules, the access controller will need to find out who are the owners of the object element that the user requests. As discussed in the previous section, the owner of the XML elements are recorded in a special column called "owner column". The access controller will perform a simple query to the RDB and find out the distinct owners of the object element.
Based on the above information, the access controller can now check from the access rules table to find out all the relevant rules applicable to the user or groups and object elements and from the relevant owners.

To apply the access rules to the XML element, the access controller needs to find out the elements user can read or write. This is called access filter. From earlier discussion we know that each XML element instance can be uniquely identified by its positional IDs. So to generate the access filter, we can actually find out what are the positional ID values for the XML elements. The access controller will use an SQL constructor to construct the SQL statements according to the condition stated in the access rules. It will then searches the equivalent database of an XML document using the SQL statements and find out all the XML element positional ID values that the user can access. Since positional ID values are numbers, they do not take much space in the memory.

The SQL constructor uses the mapping information between XML and RDB generated from the transformation stage to query the RDB. The mapping information stores the mapping between the XML element name and the table and column name in the RDB.
Finally the result set is generated from the RDB, but before constructing the XML tree structure, the system will first check these elements against the access filter to see whether they are allowed to show. After building the XML tree structure, a render agent is called to render the results into XML or HTML views and present to user.

Algorithm 3.2 Request Processing Algorithm

1. Input Request R(sub ,obj ,mode) //obj is input in term of Xpath
2. Call PROCEDURE ACM (R)
3. SQL = PROCEDURE XtoSQL(Xpath)
4. result_set = PROCEDURE SEM(SQL)
5. filtered_result_set = PROCEDURE ACF(result_set)
6. XML Document = PROCEDURE XSA(filtered_result_set)
7. Return XML Document to the user.

This is how Algorithm 3.2 works.

i) At first authorization request is passed to a program Access control module (ACM) as input in the term R = {sub, obj, mode}. The ACM makes a set of decisions based on authorization rules and policies in RDB with the help of XML-database mapping table.

ii) The ACM passes the Xpath (that has taken as obj in R) as argument to another module XtoSQL for transferring Xpath to SQL. XtoSQL returns SQL code as a string back to the ACM. The ACM then calls SQL execution module (SEM). SEM make query to Relational Database where the XML was mapped to produce a result set which is sent to the ACM. The ACM then send the produced result set to the access control filter module (ACF).

iii) ACF takes result set as argument and generates filtered result set and this ACF calls another program XML structure assembler (XSA) with argument filtered result set.

iv) XSA forms the required XML document or document portion and send it to the user.

3.5 Graphical front-end interface

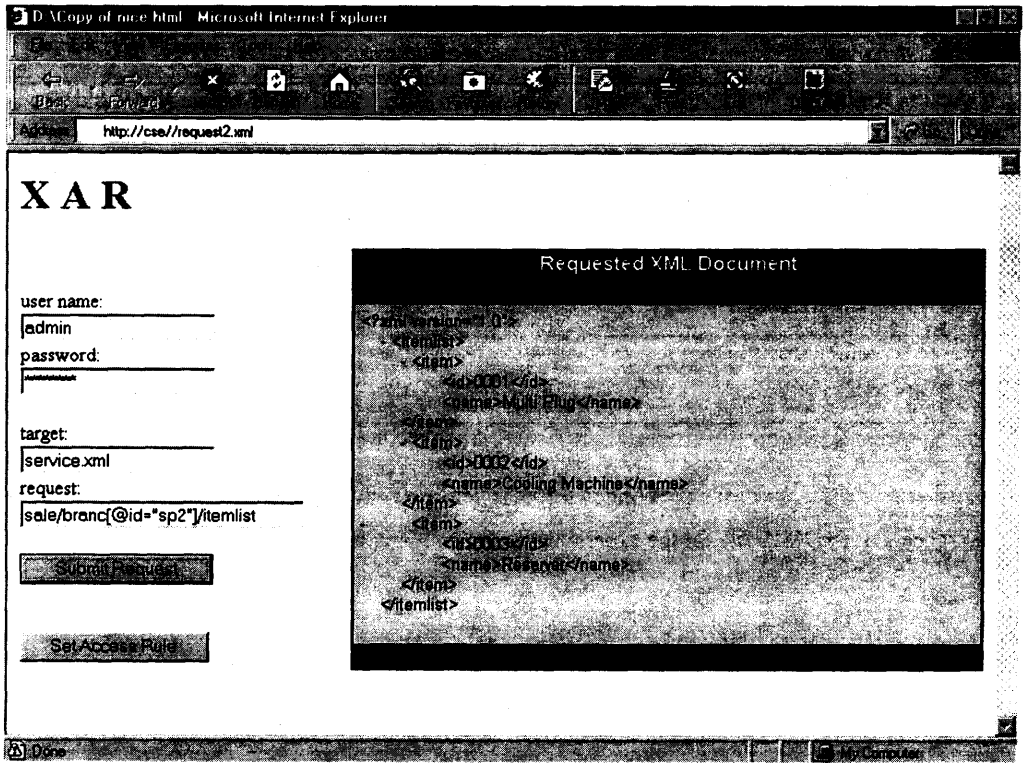

Figure 4: Request submission on XAR

We have provided a front-end graphical interface from which an administrator will be able to set the access rules to XML documents. This interface will also be available to users who have been given ownership to the elements of XML document, so that they can set the access rule for the elements they have own. More over This interface will be available in the Intra or Internet. Thus access policy can be given from remote places. Figure 4 shows a user request for a target XML document.

4.0 Study of performance of XAR

In our study of performance, we study XARs performance along with two versions of existing XML access control system and show their comparison through chart. The two systems are the DOM implementation and SAX implementation of Damiani's model [2]. The DOM implementation is the original system by Damiani, et al. [2] that requires a DOM structure to be built in memory for the XML file. The SAX implementation attempts to minimize the memory resource consumption of the DOM-based approach by using the Simple API for XML (SAX) parser.

The experiments compare the response time and memory usage between XAR and Damiani's model (DOM and SAX implementation) for different file size and for users having different access rights. The users' access rights and the portion they can view as percentage of the whole XML file size are shown in Table 1.

Table 1: User Access Rights

User	Access Right	% of whole XML file size
User1	Can only view element "issuenumber"	5%
User2	Can view all elements except element "description"	50%
User3	Can view all elements	100%

We have performed our experiments on Windows 2000 platform, with Pentium III 1.7GHz CPU and 256M RAM. The schema of the XML file for the experiment is shown in Figure 5.

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
 <xsd:element name="MyRecord">
  <xsd:complexType>
   <xsd:sequence maxOccurs="unbounded">
    <xsd:element name="article">
     <xsd:complexType>
      <xsd:sequence>
       <xsd:element name="issuenumber" type="xsd:string" />
       <xsd:element name="title" type="xsd:string" />
       <xsd:element name="initPage" type="xsd:string" />
       <xsd:element name="endPage" type="xsd:string" />
       <xsd:element name="authors">
        <xsd:complexType>
         <xsd:sequence maxOccurs="unbounded">
          <xsd:element name="author" type="xsd:string" />
         </xsd:sequence>
        </xsd:complexType>
       </xsd:element>
       <xsd:element name="description" type="xsd:string" />
      </xsd:sequence>
     </xsd:complexType>
    </xsd:element>
   </xsd:sequence>
  </xsd:complexType>
 </xsd:element>
</xsd:schema>
```

Figure 5: XML Schema of the XML Files for Comparison Experiment

When a User requests for the whole XML file, due to their different access rights, they will get different views of the XML file. The results of the response time are shown by chart in Figure 6.

In Figure 6, the vertical axis represents the response time in seconds while the horizontal axis represents the file size in megabytes. The DOM implementation of Damiani's model runs out of memory when the file size is larger than 4 megabytes.

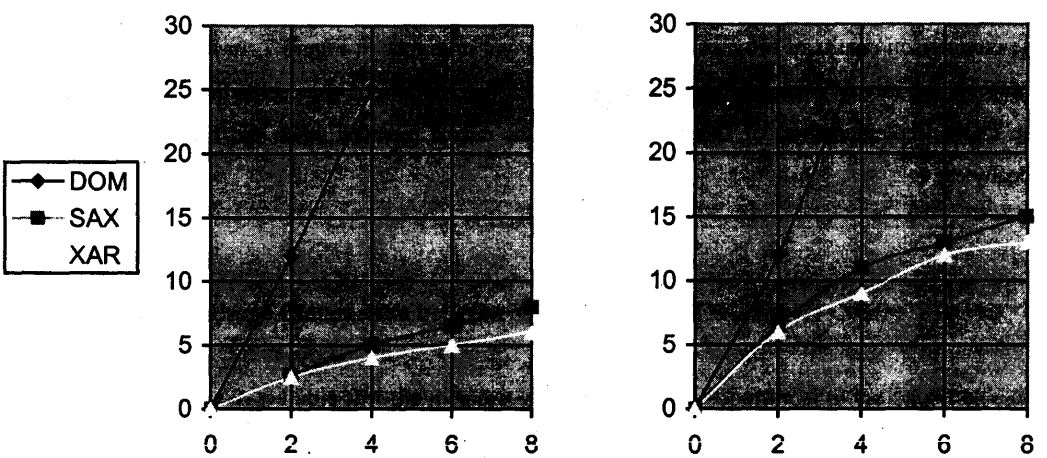

Figure 6: Response Time Comparison between XAR and Other XML Access Control Systems

of the DOM implementation. However as XAR needs more memory to construct the XML tree from the data in relational database tables, it needs more memory than the SAX implementation model.

5.0 Conclusion

In this paper an access control model for XML documents using relational database system XAR has been presented. XAR protects the XML documents using ownership access control system. This system provides flexible and fine-grained access control for XML documents so that users can get their granted portion easily and safely and at the same time the non granted data is kept secured. The comparative performance of XAR to some other traditional XML access control system has also shown in this paper and it is observed that XAR has better response time and efficient memory utilization. This model leaves space for further work. The model can be extended to role-based authorization and the performance optimizations can be investigated.

References

[1] Web Consortium (W3C) recommendation.In http://www.xml.org ,1998

[2] E. Damiani, S. Vimercati, S. Paraboschi and P.Samarati. Design and implementation of an access control processor for xml documents. In Proceeding of the 1999 WWW conference (http://www8.org/fullpaper.html), 1999.

[3] E. Bertino, M. Braun, S. Castano, E. Ferrari, M. Mesiti. AuthorX: A Java-Based System for XML Data Protection, In Proceedings of the 14th Annual IFIP WG 11.3 Working Conference on Database Security, August 2000.

[4] World Wide Web Consortium (W3C). XML Schema Part 1: Structures. In http://www.w3.org/TR/2001/REC-xmlschema-1-s20010502, 2001.

[5] http://www.w3.org/TR/xmlschema-2.

[6] E. Damiani, S. Vimercati, S. Paraboschi and P.Samarati A fine-graned access control system XML documents (http://www.xml.com).

[7] Bertino, E., C. Bettini, E. Ferrari, and P. Samarati (1996), \A Temporal Access Control Mechanism for Database Systems," IEEE Transactions on Knowledge and Data Engineering 8,1, 67-80.