Application of XML Tools for Enterprise-Wide RBAC Implementation Tasks

Ramaswamy Chandramouli
National Institute of Standards and Technology
Gaithersburg, MD 20899,USA
001-301-975-5013

chandramouli@nist.gov

ABSTRACT

The use of Extensible Markup Language (XML) and its associated APIs, for information modeling and information interchange applications is being actively explored by the research community. In this paper we develop an XML Document Type Definition (DTD) for representing the schema of a Role-based Access Control (RBAC) Model and a conforming XML document containing the actual RBAC-based access control data for a commercial banking application. Based on this DTD, the XML document and the methods in the Document Object Model (DOM) API Level 1.0 standards, we describe three application tasks related to enterprise-wide implementation of RBAC. They are: (a) implementing an RBAC model for a database application (b) implementing RBAC models with identical data on two different database servers and (c) transforming data under an RBAC model to a different, but structurally similar model like Group-based Access Control model. Other potential Access Control Service applications exploiting the capabilities of some commercial XML processors are also outlined.

1. INTRODUCTION

The Extensible Markup Language(XML) [8] had its origin as a document markup language, but the tools and API standards defined around it have greatly enhanced its potential for other applications. As a Document Markup Language, XML can be distinguished from HTML (the most widely used markup language for web applications) in the customized set of tags it provides (which can convey the semantics of the data represented) as compared to the fixed tag set of HTML. Hence an XML document can provide a logical representation of its data contents. The logical organization of the tags themselves, that are used in an XML document, are provided in a separate document called the Document Type Definition or the DTD.

Commercial software tools* called XML processors have been built to parse an XML document and create a data structure whose contents can then accessed by application programs written in languages like C++ or Java.

APIs to create, manipulate and access these data structures have been standardized. One of them is the DOM API which was issued as a W3C recommendation in October 1998 [2]. Another is the SAX API [6]. An example of a commercial XML processor based on the DOM API is IBM's XML for Java [9]. In DOM, an XML document is represented as a tree whose nodes are elements, texts, etc. An XML processor parsing an XML document generates this tree and the application program is able to manipulate the nodes of the tree through the set of DOM-provided APIs.

The ability to process an XML document contents by application programs with the help of standardized APIs opens up possibilities for XML being used for several applications besides document markup. Some of these applications are:

(a) use XML to describe the metacontent of documents on on-line resources so that they could be used by search engines,

(b) use XML to publish and exchange database contents, and

(c) use XML as a messaging format for communication between application programs. Specifically, XML is being considered as either an alternative to EDI technology or as a supporting technology for providing more flexibility for EDI-based message transfer systems.

Based on the broad categories of XML applications outlined above, the meta content description capability [item(a)] using a DTD is utilized in this paper to describe the schema of a Role-

* Certain commercial products and standards are mentioned in this paper. This does not imply recommendation or endorsement by the National Institute of Standards and Technology not does it imply that the products and standards mentioned are necessarily the best available for the purpose.

based Access Control (RBAC) Model (RBAC) [3] for a database application. The reference RBAC Model used for this purpose is the $RBAC_3$ Model (consisting of both role hierarchies and constraints) described in [7]. The actual RBAC-based access control data for the application is captured in an XML document (with the schema for $RBAC_3$ Model expressed in a DTD). The logic to implement the RBAC model in a database server for an application using these documents is described in Section 2.

The use of XML as a platform- independent data exchange format is exploited to implement RBAC models with identical data on multiple database servers. This process is described in Section 3. Further, data in an XML document that conforms to one DTD can also be captured and put into another XML document, whose structure conforms to a different DTD, using some of the methods provided in the DOM API. This capability can be utilized to implement an access control service on platforms where the supported access control model is different from RBAC, but has some structural similarities. This application is described in Section 4. A comparison of the applications described in this paper with some related work is given in Section 5. Finally, other potential applications of XML and its associated APIs for other access control service applications are outlined in the *Conclusions* section (section 6).

2. DEFINING AND IMPLEMENTING AN RBAC MODEL FOR A DATABASE APPLICATION

The database application we have chosen in this paper is a corporate banking application (which we shall call BANKDB). BANKDB contains data on (a) Customer Deposit Accounts and (b) Customer Loan Accounts as well as (c) Accounting Data related to transactions on these Customer Accounts. The application is used by Tellers, Customer_Service_Reps and Loan_ Officers to perform various transactions. It is also used by Accountants, Accounting_Managers and Internal_Auditors to post, generate and verify accounting data. The process of developing the RBAC-based access control data for this application is described below.

Tasks T1 & T2 (Role Definition and Functions/Privileges Identification) – Based on the various categories of users who will be accessing BANKDB, the participating roles and overall functionality/privileges required for each role are defined as follows:

(a) Teller – Input and Modify transactions against Customer Deposit Accounts.
(b) Customer_Service_Rep – In addition to the functionality for the Teller Role, create and delete Customer Deposit Accounts.
(c) Loan_Officer – Create and Modify status of Loan Accounts
(d) Accountant – Input all bank business transactions and generate General Ledger Reports.
(e) Accounting_Manager – In addition to the Accountant functions, the ability to modify Ledger Posting Rules
(f) Internal_Auditor – Verify all Transactions and Ledger Posting Rules.
(g) Branch_Manager – Ability to perform any of the functions of other roles in times of emergency and to View all transactions, Account Statuses and Validation Flags.

Task T3 (Role Structural Relationships) - Based on the intended functionality and the consequent privilege assignments required for each role, a structural relationship emerges among the roles. This relationship is shown through the Role Graph of Figure 2.1. In this graph, roles higher in a hierarchical chain have associated with them more privileges than the ones lower in the chain. The privilege set for any two roles which are not part of the same chain may be disjoint.

Task T4 (Formulation of Constraints) -

(a) The maximum number of users that can be assigned to Bank_Manager and Internal_Auditor roles is ONE.

(b) The following pair of roles cannot be assigned to the same user (Static Separation of Duty (SSD) or "Membership Mutual Exclusivity"):

(1) Customer_Service_Rep and Accounting_Manger
(2) Customer_Service_Rep and Internal_Auditor
(3) Loan_Officer and Accounting_Manager
(4) Loan_Officer and Internal_Auditor
(5) Accounting_Manager and Internal_Auditor
(6) Teller and Accountant
(7) Teller and Loan_Officer
(8) Teller and Internal_Auditor
(9) Accountant and Loan_Officer
(10) Accountant and Internal_Auditor

(c) The following pair of roles cannot be activated or enabled at the same user session (Dynamic Separation of Duty (DSD) or "Activation Mutual Exclusivity"):

Customer_Service_Rep and Loan_Officer

2.1 Representing the BANKDB RBAC-based Access Control Data in an XML Document

Now our follow-on tasks after determining the RBAC-based access control data for BANKDB are to define a DTD that will represent the schema for the chosen RBAC Model (referred to in Section 1) for this application and then to capture the actual data in a conforming XML document. There are several issues we have to consider while defining a DTD for representing the schema of an RBAC Model:

(a) Expressiveness – should be able to capture the semantics of the various RBAC Model constructs.

(b) Flexibility – it would be preferable if the DTD is generic enough to be used for describing most common RBAC models, not merely the one used for BANKDB.

(c) Document-Readability – the conforming XML document is readable and hence the logic of the RBAC implementation program that parses this document is not unduly complicated.

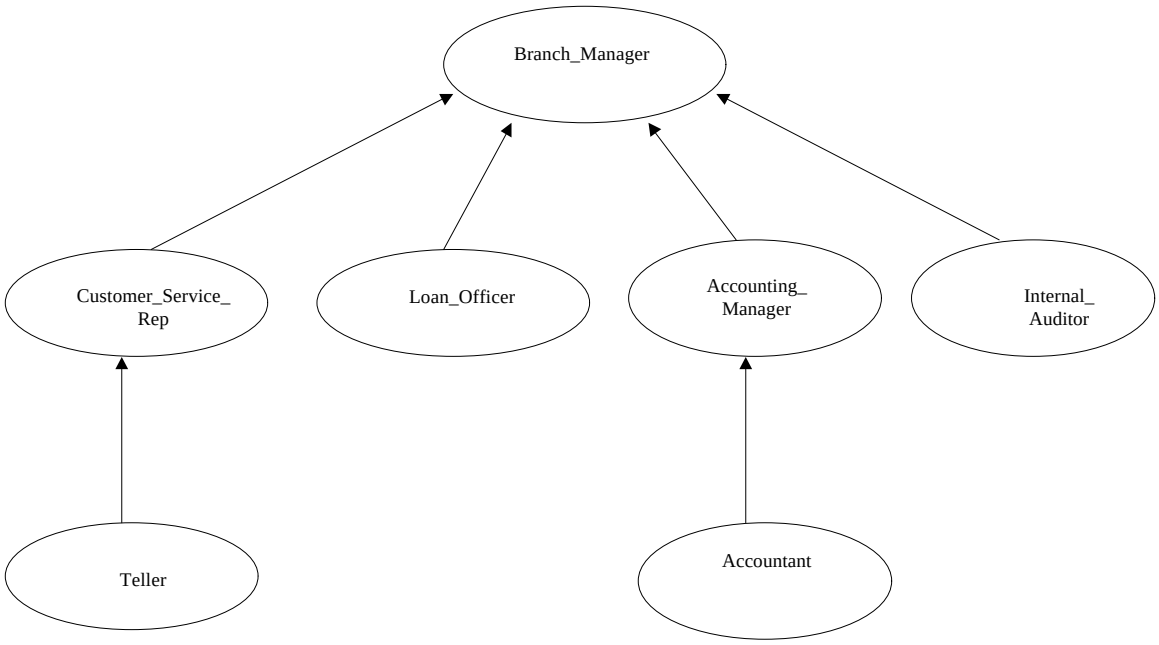

Fig 2.1 Role Graph for Bank Database Application (BANKDB)

Unfortunately, "Expressiveness" and "Document-Readability" turn out to be conflicting requirements. An expressive DTD whose elements truly reflects the semantics of all the constructs (e.g., Parent_Role, Child_Role, etc.) of the RBAC model may make the conforming XML document unreadable due to the multiple levels of nesting of various tags. Taking into consideration this trade-off, we have developed the following DTD for representing the RBAC model schema:

Listing 2.2 – RBAC.dtd

```
<!ELEMENT Role_Graph  (Application ,
                       (role )* )*>

<!ELEMENT Application  (DB_Name ,
                       Server )>
<!ELEMENT DB_Name  (#PCDATA )>
<!ELEMENT Server  (#PCDATA )>

<!ELEMENT role  (Name , Cardinality? ,
        (Parent_Role?)* ,  (Child_Role?)* ,
       (SSD_Role?)* , (DSD_Role?)* )>
<!ELEMENT Name  (#PCDATA )>
<!ELEMENT Cardinality  (#PCDATA )>
<!ELEMENT Parent_Role  (#PCDATA )>
<!ELEMENT Child_Role  (#PCDATA )>
<!ELEMENT SSD_Role  (#PCDATA )>
<!ELEMENT DSD_Role  (#PCDATA )>
```

A fragment of the XML document that conforms to the above DTD which contains RBAC-based access control data relating to BANKDB is given below:

Listing 2.3 – BANKDB_RBAC.XML (fragment)

```
<?xml version="1.0" ?>
<!DOCTYPE Role_Graph SYSTEM
      "RBAC.dtd">
<Role_Graph>
    <Application>
      <DB_Name>Bank Corporate
               Database</DB_Name>
      <Server>Solaris</Server>
    </Application>

    <role>
      <Name>Branch_Manager</Name>
      <Cardinality>1</Cardinality>
      <Child_Role>Customer_Service_Rep
                  </Child_Role>
      <Child_Role>Loan_Officer
                  </Child_Role>
     <Child_Role>Accounting_Manager
                  </Child_Role>
     <Child_Role>Internal_Auditor
                  </Child_Role>
    </role>
…….
</Role_Graph>
```

Since the above XML document was created manually using a conceptual RBAC model for BANKDB, it has to validated for conformance to the schema RBAC.dtd. Many commercial XML processors like the IBM's XML for Java [9] do perform this validating function.

2.2 Implementing the RBAC Model for BANKDB using the data in BANKDB_RBAC.XML

Let us now develop a Java program (by name RBAC_XML_ TO_DB.java) to read the data in the XML document BANKDB_ RBAC.XML The use of Java programs to parse XML documents (by invoking a XML 1.0 conformant processor) and making use of DOM API methods to extract the relevant contents has been illustrated in [5]. Our Java program needs to do many additional tasks after extraction of the needed data. The three overall tasks that our Java program has to perform follow:

Task A - Parse the XML document BANKDB_RBAC.XML and generate the internal DOM tree representation of this document.

TASK B - Navigate through the nodes of this DOM tree to extract the data representing the names of the roles, role containment relationships (parent and child roles) and role constraints by navigating to the appropriate nodes.

TASK C - Based on the semantics associated with the extracted data, generate the corresponding SQL command, to either (a) create a role **or** (b) specify a structural relationship **or** (c) a constraint involving previously created roles. Pass these SQL command strings as parameters through appropriate JDBC methods to implement them on the database server.

Repeat tasks (TASK B and TASK C) for all the relevant nodes found in the DOM tree. The complete schematic diagram depicting the above three main tasks is given in Figure 2.4.

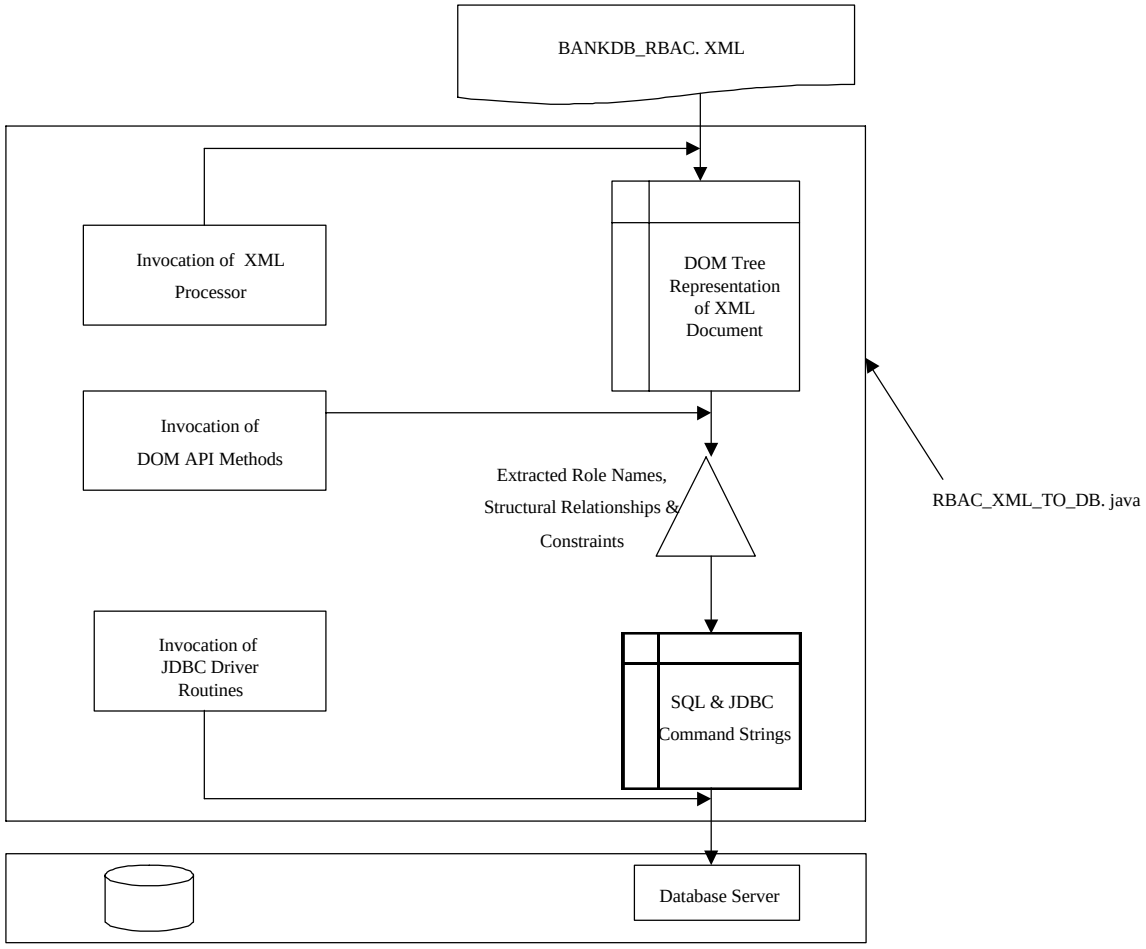

Figure 2.4 – Implementing an RBAC Model on a Database Server using data from an XML Document

.

3. IMPLEMENTING RBAC MODELS WITH IDENTICAL DATA ON MULTIPLE DATABASE SERVERS

There may be many situations in enterprise environments where identical access control requirements are needed in two different database servers. For example in our BANK environment, the Customer Deposit Accounts may be partitioned along geographical states and housed in different servers, but the access control restrictions have to be identical. Implementing RBAC models with identical access control data in two database servers (let us call DBServerA and DBServerB) cannot be accomplished using bulk data transfer utilities which are the common means for transferring data between database servers. This is due to the fact that access control data is not an application data, but a data used by the security module of the DBMS and hence stored in System (data dictionary) tables. Generally, DBMSs do not allow direct updates (except for one or two data items) to the contents of System tables. So the only way to implement RBAC models with identical data on two database servers is to extract the needed RBAC-based access control data from DBServerA, express it in a database server-neutral format and then parse that data to generate the necessary SQL commands to implement an RBAC model on DBServerB[**]. Using XML as the server-neutral format, this process translates to the following set of tasks:

TASK A - Using a Java program (let us call it DB_TO_RBAC_XML.java) which makes of JDBC access libraries and appropriate SQL commands, retrieve the RBAC-based access control data from DBServerA. Generate an XML document (let us call it as DBServerA_RBAC.XML) using that retrieved data such that it conforms to RBAC.dtd.

.

TASK B - By parsing the XML document DBServerA_RBAC.XML, generate a set of SQL commands to implement an RBAC model on DBServerB.

It should now be fairly obvious that Task B (implementing an RBAC model on a database server by processing the data provided in an XML document) is what we accomplished in Section 2. Hence the only task we need to address is TASK A.

Let us now look at the issues involved in this XML document generation. To generate any XML document from scratch, the first thing we need is to generate an internal DOM tree representation of it. In our scenario, the *structure for this DOM tree* (which represents the role's parent-child relationships, constraints etc) should conform to RBAC.dtd representing the RBAC model schema. The *contents of the DOM tree* (texts denoting role names) are to be obtained through SQL queries against DBServerA. Generating an internal DOM tree corresponding to a given DTD is called the "Validating Generation." Unfortunately, the DOM API Level 1.0 specification neither provides methods for "Validating Generation" nor for subsequently generating an actual XML document file from a valid DOM tree. However some commercial XML processors like IBM's XML for Java [9] do provide Java libraries for carrying out these functions.

Having addressed the issue of "validating generation of an XML Document," we now have all processes in place to accomplish our goal of implementing RBAC models with identical data on database servers DBServerA and DBServerB. The complete schematic diagram of all processes involved is shown in Fig 3.1.

4. USING RBAC-BASED ACCESS CONTROL DATA FOR IMPLEMENTING AN ACCESS CONTROL SERVICE BASED ON OTHER MODELS

There are many enterprise platforms (including DBMS and Operating Systems) which do not support RBAC as the access control model, but a different but nonetheless structurally similar models like Group-based access control. For example, the concept of Group may support building a hierarchical chain using Super_Groups and Sub_Groups but may not permit association of any constraints other than Group membership limit. Hence there may be a need to transform access control data in the enterprise based on the concept of roles to ones based on concepts like Groups. In other words, if we have captured this RBAC-based access control data in an XML document (with an associated DTD), then there is the requirement to convert this document into another XML document (based on a different DTD).

Within the context of our discussion, the above requirement implies that the XML document BANKDB_RBAC.XML (based on RBAC.dtd) must be transformed into one based on a DTD defined for representing the schema for a Group-based Access Control Model. An example of such a DTD is given below:

Listing 4.1 – Group_Access.dtd

```
<!ELEMENT Group_Org  (Application ,  (group)* )>

<!ELEMENT Application  (DB_Name , Server )>
<!ELEMENT DB_Name  (#PCDATA )>
<!ELEMENT Server  (#PCDATA )>

<!ELEMENT group (Name , Membership_Limit? ,Super_Group?, (Sub_Group?)* )>
<!ELEMENT Name  (#PCDATA )>
<!ELEMENT Membership_Limit  (#PCDATA )>
<!ELEMENT Super_Group(#PCDATA )>
<!ELEMENT Sub_Group  (#PCDATA )>
```

On comparing the above DTD with the DTD (RBAC.dtd) of our source XML document (BANKDB_ RBAC.XML) we find the following node mappings are required.

[**] The SQL commands to implement RBAC models on DBServerA and DBServerB will not be identical if these servers are from different DBMS vendors. For a comparison refer [1].

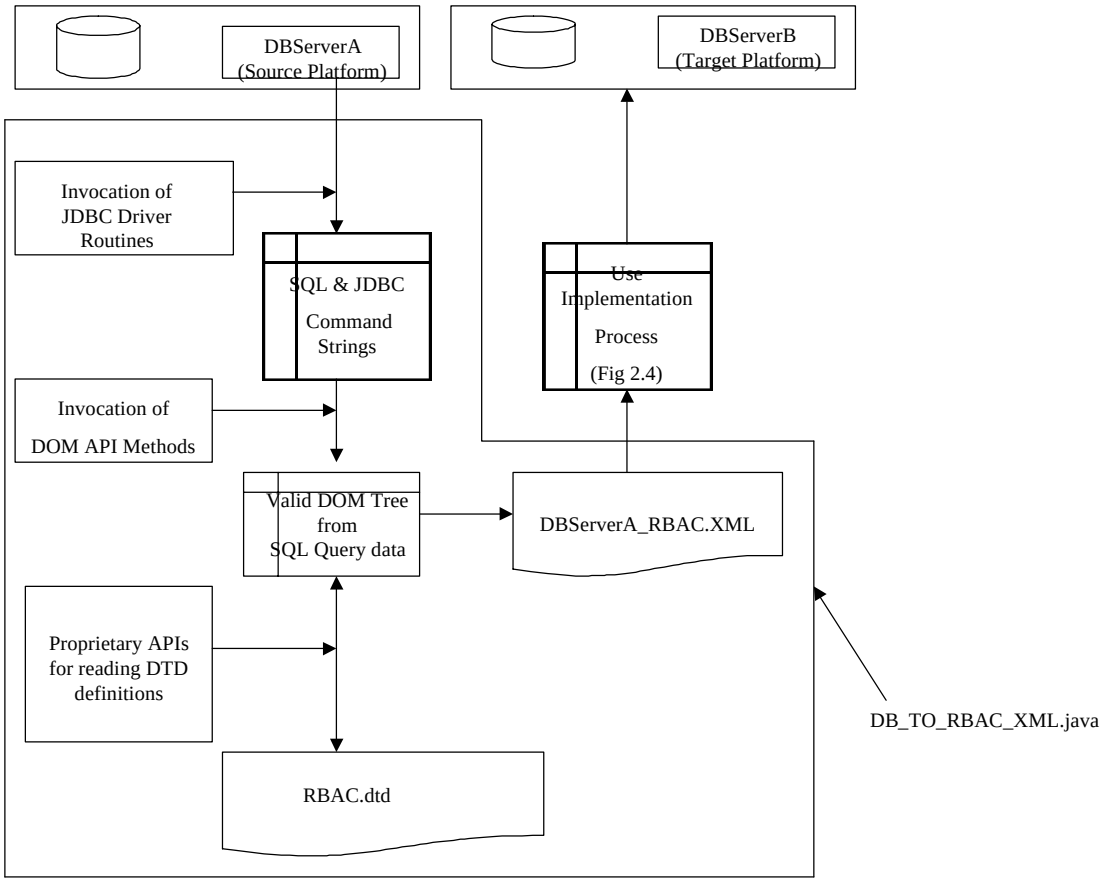

Fig 3.1 – Implementing RBAC Models with Identical Access Control Data on two Database Servers

RBAC. dtd	**Group_Access. dtd**
role	group
Parent_Role	Super_Group
Child_Role	Sub_Group
SSD_Role	NONE
DSD_Role	NONE

Table 4.2 – DTD Element Mappings

Now these node mappings themselves can be expressed in an XML document using patterns as described in [5]. With the creation of this XML document (which we shall as RBAC_ TO_Group_Mappings.XML) we are now in a position to describe the sequence of tasks needed for XML document conversion.

TASK A - Parse the source XML document BANKDB_RBAC. XML (based RBAC.dtd) and generate the internal DOM tree of this document.

TASK B - Parse the XML document (RBAC_TO_Group_Mappings.XML) and create a DOM tree for the mappings.

TASK C - Using the DOM tree data created from TASK A (which contain element names – like role, Parent_Role etc., and embedded text data – Teller, Accountant etc.,) and conversion mappings obtained from TASK B, create the converted DOM tree. Use this converted DOM tree to generate the target XML document.

We shall now illustrate the process of converting an element (an element is one type of node) in the DOM tree of the source XML document to an equivalent element in the DOM tree of target XML document using some sample methods from the DOM API.

Supposing the DOM tree objects corresponding to the source XML document, XML document containing the conversion patterns, and the target XML document are *sourcedoc*, *patterndoc* and *targetdoc* respectively. Also let us assume that initially the *sourcedoc* and *targetdoc* are identical. Now the problem is to replace elements in *targetdoc* for which element mappings are found in *patterndoc*. To carry out this process it is essential to access a node in the *sourcedoc* and verify whether it is an element node. This can be done using the following method invocations.

```
/* Obtain an object of type Node */

Node sourcenode = sourcedoc.getDocumentElement(). GetFirstChild()

/* Check whether it is an ELEMENT node */

If (sourcenode.getNodeType() ==
      Node.ELEMENT_NODE
```

Let us now call the objects in *patterndoc* that denote the pair of mapped elements as *FromNode* & *ToNode*. Now the next step in our process is to verify whether the extracted element in the source document matches with a *FromNode* and if so create a new node with a name equal to that of the *ToNode*.

```
If (sourcenode.getNodeName(). equals
(FromNode.getNodeName()) {

/* create a new node and set its name */

Node NewNode = new Node();
NewNode.setNodeName() =
    ToNode.getNodeName(); }
```

The last step in this process is to retrieve a node in the target DOM tree (which needs to be replaced –which now has the same name as the node in the source DOM tree) called *OldNode*, navigate to its parent and replace the *OldNode* with the *NewNode*.

```
ParentNode = OldNode.getParent();
ParentNode.replaceChild(NewNode, OldNode);
```

The complete schematic diagram of the XML document conversion logic is given in Fig 4.3.

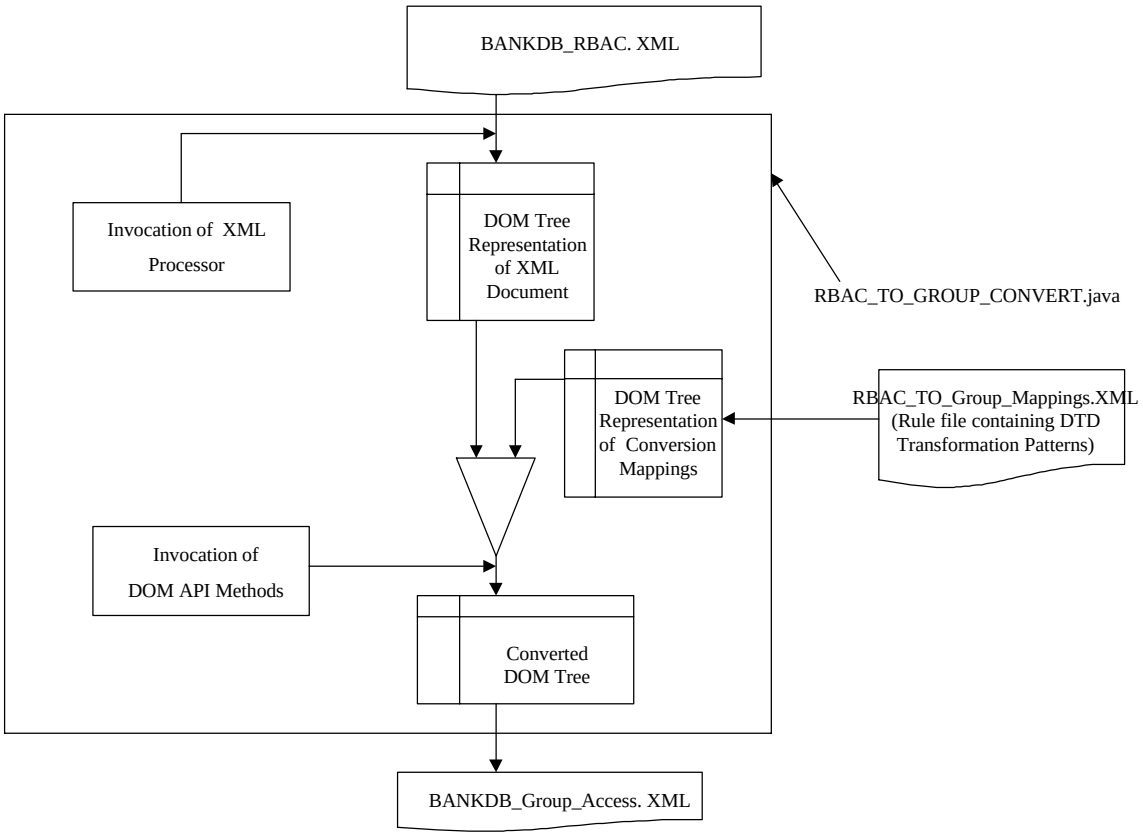

Figure 4.3 Transforming an XML Document containing RBAC-based Access Control Data into an XML Document containing Data for Group-based Access

5. COMPARISON WITH RELATED WORK

The work that can compared with the XML-based implementation techniques outlined in this paper is the MIX Project [4]. The MIX Project has focused on mapping XML DTDs to relational schemas as well as deriving candidate DTDs for given relational schemas. The work outlined here involves much more sophisticated processing requirements (using DOM API and some proprietary methods) than the simple schema mapping techniques outlined in MIX. This is due to the reasons already stated earlier – that RBAC-based access control data is not an application data located under a specified relation but is distributed over several DBMS system tables which cannot be directly updated.

6. CONCLUSIONS

In this paper we have illustrated the capabilities of XML processors and some of the methods in its associated APIs to perform three common enterprise-wide tasks relating to implementation of access control service based on RBAC. Several extensions to the application ideas outlined in this paper are possible. For example, we could use the XML document conversion ideas discussed in Section 4 to extract access control data from several platforms and represent the access control data for the entire enterprise through a common model. Similarly access control data under an Enterprise Model can be translated to ones that are native to the platforms.

7. REFERENCES

[1] Ramaswamy Chandramouli and Ravi Sandhu, " Role-Based Access Control Features in Commercial Database Management Systems," In Proceedings of the 21st National Information Systems Security Conference, pages 503-511, Arlington, VA, (October 5-8, 1998).

[2] Document Object Model – Level –1 Recommendations. In http://www.w3.org/TR/REC-DOM-Level-1(October 1998).

[3] David Ferraiolo, Janet Cugini and Richard Kuhn, "Role-based Access Control (RBAC): Features and motivations," In Proceedings of 11th Annual Computer Security Applications Conference, pages 241-48, New Orleans, LA, (December 11-15, 1995).

[4] Mediation of Information using XML. In http://www.npaci.edu/DICE/MIX, San Diego Super Computer Center, La Jolla, CA (1999).

[5] "XML and Java – Developing Web Applications" by Hiroshi Maruyama, Kent Tamura and Naohiko Uramoto, Addison-Wesley, Reading, MA, (1999).

[6] http://www.megginson.com/SAX

[7] Ravi S. Sandhu, Edward J.Coyne, Hal L. Feinstein and Charles E. Youman, "Role based access control models," IEEE Computer, 29(2): 38-47, (February 1996).

[8] The Extensible Markup Language, Version 1.0. In http://www.w3.org/TR/REC-xml, (February 1998).

[9] http://www.ibm.com/xml