EXTENSIBLE ACCESS CONTROL MODEL FOR XML DOCUMENT COLLECTIONS

Goran Sladić, Branko Milosavljević and Zora Konjović
Faculty of Technical Sciences, University of Novi Sad, Fruškogorska 11, Novi Sad, Serbia

Keywords: XML, RBAC, Access control, Information security.

Abstract: This paper presents the XXACF (eXtensible Role-Based XML Access Control Framework) framework for controlling access to XML documents in different environments. The proposed access control definition language and the corresponding software architecture are described. The framework enables defining access control policies on different priority and granularity levels. The XXACF enables the enforcement of access control for different operations on XML documents, as well as different ways of access control enforcement for the same operation. This framework's configurability facilitates customization of particular implementations according to specific needs. Extensibility of XXACF framework is achieved by the possibility of extending the core functionality for specific requirements and also the addition of the new modules for context-sensitive access control.

1 INTRODUCTION

Access control is only one aspect of a comprehensive computer security solution, but also one of its most important segments. It provides confidentiality and integrity of information.

In the role-based access control (RBAC) model, access to resources of a system is based on the role of a user in the system (Ferraiolo et. al., 2001; Ferraiolo et. al., 2003). Basic RBAC model comprises the following entities: *users, roles* and *permissions*, where permissions are composed of *operations* applied to *objects*. In RBAC, permissions are associated with roles, and users are made members of roles, (Ferraiolo et. al., 2001; Ferraiolo et. al., 2003).

The growth of XML as a format for data modeling and interchange arises the issue of access control to XML documents. An XML document may contain data of different level of accessibility.

The Extensible XML Role-Based Access Control Framework (XXACF) provides the means of defining access control policies and access control enforcement based on users' roles. Access control policies in XXACF may be defined in different priority and granularity levels and they may be content depended, thus facilitating efficient management of access control. The access control policy in XXACF may be separately defined for each operation on an XML document.

The framework itself is modularly organized in the manner which provides a simple update or upgrade of the existing modules, as well as the inclusion of new ones. The extensibility of the XXACF is achieved by extending the core functionality according to the specific requirements, as well as including new modules for the purpose of context-sensitive access control enforcement. The concept of context-sensitive access control enables customization of access control policies depending on the environment where XXACF is being used. Therefore, XXACF can be deployed in various environments.

XXACF is implemented in Java EE technology and based on open source components. It is used as a module of the application server in a multi-tier software architecture. The prototype implementation, used for controlling access to XML documents, is tested in a complex real-life workflow system. The additional modules for context-sensitive access control, as well as modules for integration with the workflow system were implemented in order to satisfy specific requirements of the access control for the business process implemented through the workflow system.

2 RELATED WORK

X-RBAC (Bhatti et. al., 2004) is a system for controlling access to XML documents in web service-based systems. The access control in this system is based on the RBAC model. The access control policies depend on the user's session context and the content of the documents being accessed (Bhatti et. al., 2004). Depending on document content, the access control policies can be specified on four granularity levels: conceptual level, document's schema level, document instance level and the document's element level (Bhatti et. al., 2004). However, policies that deny access can not be specified. The supported operations are *reading*, *writing*, *navigate* (reading the referenced data) and *all* (supports all three previous operations). The propagation level down through hierarchy can be defined for each access control policy. The three levels of propagation are supported: (a) without propagation, (b) first level propagation and (c) cascade propagation (Bhatti et. al., 2003).

In papers (Botha and Eloff, 2001a; Botha and Eloff, 2001b) access control for XML documents in a workflow environment is presented. The access control policies of the workflow system are based on the RBAC model. There are two types of access control policies (Botha and Eloff, 2001b): policies which grant access (operation) and policies which deny access (operation) on the object. The operations that are supported are as follows: *read*, *edit*, *add* and *delete* (Botha and Eloff, 2001b). Each process in the workflow system implies the manipulation of certain number of XML documents. The definition of the given process defines which roles can execute that process, i.e. its sub-processes. It is possible to create access control policies not only on the document level but also on the document fragment level (Botha and Eloff, 2001a).

XML ACP (Access Control Processor) (Damiani et. al., 2000) represents a system for access control for XML documents in web-based applications. The system works as a plug-in to the existing web server technology (Damiani et. al. 2000). The access control is possible on several levels of granularity: DTD level, document instance level, specific document's element/attribute level. The access control policies can be classified in eight priority groups. The access control policies propagation down through the hierarchy is supported. The policies can grant or deny access (Damiani et. al., 2002). In access control policy definitions, user identification is based on the user ID or the user group ID and/or the location (computer or computer network) ID being the origin of the request to access a document (Damiani et. al., 2002).

Author-X system (Bertino et. al., 2001; Bertino and Ferrari, 2002) provides for defining access control policies on different granularity levels, which can grant or deny the access. The controlled propagation of access control policies is provided, where policies defined for document or DTD can be applied on the other semantically related documents or DTDs on different granularity levels (Bertino and Ferrari, 2002). There are two working modes: *pull* and *push* (Bertino et. al., 2001). In the pull mode, the user requests explicitly the access to the document. After the reception of the request, the X-Access forms a document that will contain only the data visible for the user, i.e. the data that the user can change. After the change has been done, the user sends the changed content to the system and the system verifies whether the changed content is in accordance with security rights (Bertino et. al., 2001). In the push mode, the system periodically sends documents to all users. Although the same document is sent to all users, the specified security rights are enforced by encrypting parts of the document with different keys for different policies. Each user possesses the keys which are available to him or her according to the specified security rights (Bertino et. al., 2001).

Table 1 shows comparation of the XML access control systems described in this chapter and the XXACF system.

3 XXACF DATA MODEL

3.1 Roles

The roles are defined in an XML document conforming to the XML schema presented in Figure 1. The name must be specified for each role (the `name` attribute of the `<role>` element). If the value of the `abstract` attribute of the `<role>` element is "*true*", then that role can not be directly assigned to the user, i.e. it's a connector role. The content of the `<description>` element represents a short description of the role. For each role, it is possible to specify its parent roles (`<parent_roles>` element) and to establish the role hierarchy with multiple inheritance (Ferraiolo et. al., 2003).

Table 1: Comparation of XML access control systems.

	usage	*AC model*	*granularity levels*	*policy propagation*	*cont-sens*	*operations*	*extensible*
X-RBAC	web-services	RBAC based on credentials	conceptual, schema, instance, element	no propagation, one level, cascade down	YES	prune read, write, navigate	NO
XAC WFMS	WFMS	RBAC	schema, instance, fragment	no propagation, cascade down	NO	prune read, add, delete, edit	NO
XML ACP	web	users, groups, computers, domains	schema, instance, element, attribute	no propagation, cascade down	NO	prune read	NO
Author-X	general	credentials	schema, instance, element	no propagation, one level, cascade down	NO	prune & encrypted read, add, delete, edit	NO
XXACF	general	RBAC	schema, instance, fragment	no propagation, cascade down or up 1-N levels	YES	prune, encrypted & dummy read, add, delete, edit, provisional	YES

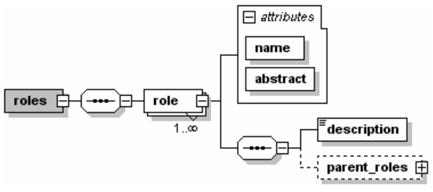

Figure 1: XML schema of the roles.

3.2 Users

The user is identified by the `id` attribute of the `<user>` element (see Figure 2). The user's personal data is contained in the `<personal_info>` element. The entities which comprise user's personal data are defined according to the system requirements. The XXACF is designed and implemented to be independent of the entities that represent user's personal data. The user's data regarding authentication and authorization are defined in the `<principals>` element. Each user is allowed to own several `<principal>` elements. The `<principal>` element, identified by `id` attribute, contains authentication data (`<authentication>` element) and roles (`<roles>` element) which will be assigned to the user when he or she logs in using the authentication data of that principal. That way it is possible that the user owns several user accounts, with specific roles assigned to each user account.

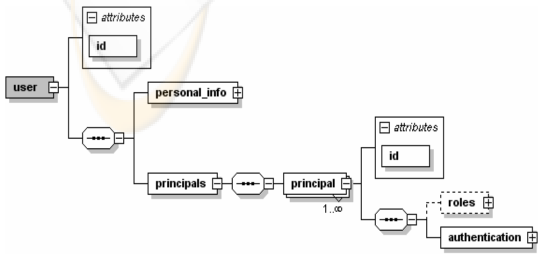

Figure 2: XML schema of the users.

3.3 Access Control Policies

The XML schema in Figure 3 defines the structure of access control policies, which is based on policies described in (Hada and Kudo, 2002) supporting provisional operations and policy conditions. Access control policies described in (Hada and Kudo, 2002) are extended in order to support role hierarchies, to provide priorities of the access control policies as defined in (Damiani et. al., 2000; Damiani et. al, 2002) and to support arbitrary operations for which policies are defined. Each access control policy is defined by the `<policy>` element to which a unique identifier (attribute `id`) is assigned. The policy permits or denies (`<permission>`) the subject to execute the operation (`<operation>`) on the object (`<object>`) (Ferraiolo et. al., 2003). Since access control policies are based on the RBAC model, the subject, for which the policy is defined, represents the role. The object of the access control policy is a document schema or a document instance identified by its unique ID. If the policy is defined only for the particular document (schema) fragment, it is necessary to specify the XPath expression which selects that fragment. Also, by using XPath expressions with conditions, content depended object (policies) may be specified. Apart from the main operation for which the policy is defined, there may be some provisional operations (Hada and Kudo, 2002). If the condition (`<condition>`) is satisfied, the access control policy will be applied. Contrary, it will not be the case. Each condition is consist of the name of a logical operation (*not*, *and*, *or* and *xor*) (attribute `operation`) and a subconditions and/or a predicates (`<predicate>` element). By predicate is meant function which return value is *true* or *false*. Value calculated by applying a specified logical operation on return values of subconditions and predicates represents the return value of a condition. The functionality of the each predicate should be properly implemented. This gives possibility for implementation of a specific access control which is not supported through the RBAC model or is much efficient to implement in this manner than using presented policies without conditions (i.e. insertion of a digital signature into document is allowed if signature is valid and generated using the key of the certain user). By using a predicate which return value depends on the state of an environment (i.e. access to document depends on current state of the process to which document belongs to), context-sensitive access control (Ferraiolo et. al., 2003) is possible.

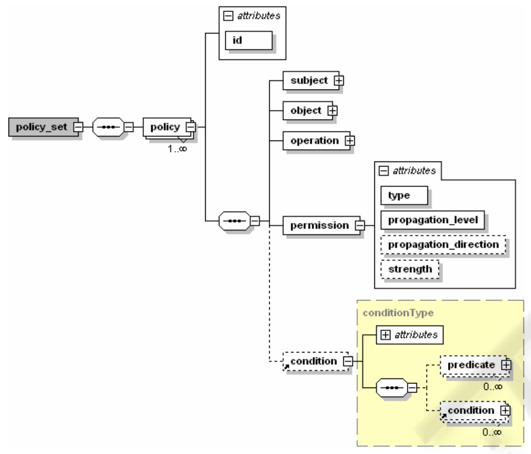

Figure 3: XML schema of access control policies.

The permission of policy is described by the `<permission>` element showed in Figure 3. The permission can be dual, to grant access (the value of the `type` attribute is "*grant*") or deny access (the value of the `type` attribute is "*deny*"). In order to avoid specifying explicitly the policy for each entity, the propagation of policies is enabled, starting from the entity specified by the `<object>` element downwards or upwards the hierarchy. The attribute `propagation_direction` defines the propagation direction. If the attribute value is "*down*", the propagation is directed from the specified object downwards along the hierarchy. On the other hand, if the value is "*up*", it is directed from the specified object upwards. The level of propagation, i.e. the maximum number of hierarchy levels the propagation is performed, is specified by the attribute `propagation_level`. The propagation level can be unlimited, a certain number of levels or with no propagation at all. Access control policy strength is defined by the `strength` attribute as defined in (Damiani et. al., 2000; Damiani et. al., 2002). Values of the `strength` attribute can be "*normal*", "*hard*" and "*soft*". The value "*normal*" can be specified if the object of the access control policy is the document schema or the document instance, "*hard*" value is specified only for the document schema and "*soft*" value if the object is the document instance.

Access control policies defined for a descendant role have a higher priority than the policies defined for the ascendant role. Priority of the access control policies defined for the same role is determined according to the object they are related to (document schema or document instance), to the strength of the policy and also to the propagation level of the policy. On the basis of these elements it is possible to define eight priority levels (the priority is descending from the first to the last): (1) the policy defined for the document schema, with no propagation, strength is "*hard*", (2) the policy defined for the document schema, with propagation, strength is "*hard*", (3) the policy defined for the document instance, with no propagation, strength is "*normal*", (4) the policy defined for the document instance, with propagation, strength is "*normal*", (5) the policy defined for the document schema, with no propagation, strength is "*normal*", (6) the policy defined for the document schema, with propagation, strength is "*normal*", (7) the policy defined for the document instance, with no propagation, strength is "*soft*" and (8) the policy defined for the document instance, with propagation, strength is "*soft*".

4 ACCESS CONTROL IN XXACF

The process of access control is performed within four steps: (1) a selection of the applicable access control policies, (2) marking document nodes, (3) conflict resolution and (4) execution of the requested operation. Each of these steps is implemented within one or more modules of the system. Hence, a change in the algorithm of a particular step or the whole access control procedure is performed by changing or adding a certain module(s).

4.1 Selection of the Applicable Access Control Policies

When a document is being accessed in order to execute a certain operation, access control policies defined for that document and its schema are loaded from repositories. The access control policies being loaded must be defined for the specified operation and for the roles assigned to the user who performs the operation on that document.

The system tests the condition of usage for each loaded access control policy. If condition is satisfied, the access control policy will be applied; otherwise it will not be applied.

4.2 Marking Document Nodes

Marking the document nodes is performed on the DOM (Document Object Model) model of the document. In order to provide the means for access control in XXACF, the DOM model is extended with the additional functionalities. The diagram in Figure 4 shows the classes which represent the marking of a DOM tree.

Each implementation of the `XACNode` interface contains the `MarkMap` object which contains all access control policies applied on that node (Figure 4 shows the example for the `XACElementImpl`). The name of the role for which the access control policy is defined is used as the key of the hash table contained in the `MarkMap` class. Hash table values are the instances of the `MarkMatrix` class which contain access control policies defined for the given role and applied on that node.

The `MarkMatrix` distributes access control policies on the one of `ArrayList` lists of the `matrix` matrix. The row index depends on whether the access control policy grants or denies access to that node. The column index represents the priority level of the access control policy. Therefore, `matrix` has two rows and eight columns.

Each of these lists contains instances of `MarkItem`. A `MarkItem` contains the access control policy and the distance from the root node selected by the given policy to the node to which belongs that `MarkItem` instance.

For each access control policy, the class `Marker` determines which nodes of the tree are selected by the object of the policy and applies the policy on these nodes.

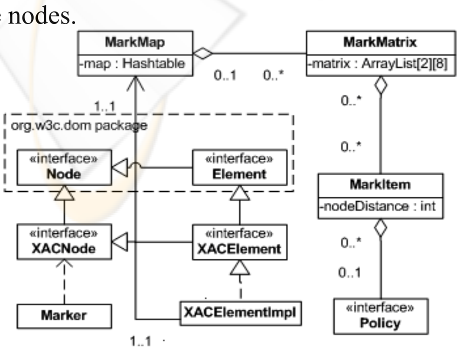

Figure 4: Classes for marking.

4.3 Conflict Resolution

After marking is finished, it is necessary to resolve conflicts (if any) and determine the final access control policies which will be applied on the nodes. Upon conflict resolution there may be several final access control policies for each node, where all final access control policies either grant or deny the access.

The first activity in selection of final access control policies is conflict resolution according to the "*most specific subject takes precedence*" (MSSTP) principle (see Figure 5). According to this principle, access control policies defined for the roles on the lower hierarchical level have higher priority than the ones defined for the roles on the higher level. For each role it is checked whether there are policies defined for that role. If the policies defined for that role exist, they are selected. On the other hand, if they don't exist, the role hierarchy is recursively traversed for policies.

When access control policies of a role are added to the list of selected policies it is checked whether the list already contains policies defined for one of the ascendant roles. If there are policies defined for any of the ascendant roles, they are removed from the list. That can happen if a role A is an ascendant to roles B and C, and there are no defined policies for B, but these policies exist for roles C and A. Since there were no policies for role B, the policies of role A are applied according to the MSSTP principle. If there are policies defined for role C (which override policies defined for role A), the policies of role A are not used. But, if the policies of the role A are already in the list of the selected policies, they must be removed in order to maintain the MSSTP principle of conflict resolution.

Besides, while adding policies defined for a certain role to the list, it is necessary to verify if there are any policies of descendant roles in the list. If they exist, given policies will not be added to the list. That can occur if one role A is and ascendant to roles B and C and there are defined policies for roles A and B, while there are no policies defined for role C. The conflict resolution procedure takes the policies defined for role B. Since there are no policies for role C, policies of its ascendant role (A) are taken. But, since policies of role B are already in the list of selected policies, policies of role A will not be added in order to maintain the MSSTP conflict resolution principle.

The next activity is conflict resolution using "*most specific object takes precedence*" (MSOTP) principle (see Figure 5). It is performed on the policies selected in the previous activity. In this step, the policies whose object is nearest to the node on which the policy is applied are selected. The distances between the root node selected by the object of the policy and the node on which access control policy is applied are compared.

The result of previously described activities is an instance of `MarkMatrix` class which contains access control policies (of all selected roles) arranged according to priority levels. The policies from the first highest non empty level of matrix are retrieved.

If, in the selected priority level, there are access control policies that grant and deny access, the defined conflict resolution is verified. In case that defined conflict resolution uses "*denial takes precedence*" (DTP) principle, the final access control policies are the ones that deny access. In case that the applied principle is "*grant takes precedence*", the final access control policies are the ones that grant access.

In case that in the selected priority level all access control policies either grant or deny access, the final access control policies grant access, or deny it.

4.4 Operation Execution

XXACF supports the following operations on XML documents: (a) updating documents (inserting new nodes, deleting and changing (replacement) of the existing ones) and (b) reading documents. The reading operation is implemented in three different ways: pruned reading, dummy reading and encrypted reading. Apart from the already implemented document operations, new operations may be added by implementing an XXACF module and registering it within the XXACF framework.

4.4.1 Update Operations

A new subtree to be inserted into the document tree is temporarily inserted. Then, the selected access control policies are applied on the whole tree and the conflict resolution process is performed. If the final access control policies on each node of the inserted subtree grant the access, the insertion is allowed. If the insertion of any node of the inserted sub tree is not allowed, the subtree is removed from the document tree because its insertion is not allowed.

Analogously to the insertion operation, a delete operation of the subtree is allowed if a deletion of all nodes of the subtree is allowed, i.e. if the final access control policies of each node of subtree permit this operation.

For the replacement of some subtree by another subtree, it is necessary that access control policies allow replacement of the nodes, i.e. that it is allowed to replace one of the nodes and that replacement with the given node is allowed. Hence, two sets of access control policies have to be defined for the operation of replacing a subtree. The first set defines the precondition (whether the node can be replaced). The second set defines the postcondition (whether the replacement with a given node is allowed).

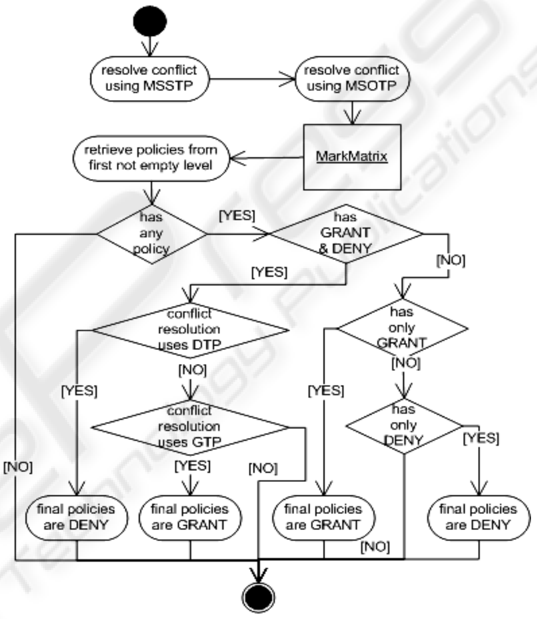

Figure 5: Conflict resolution.

4.4.2 Reading Operations

The XXACF currently supports three types of reading operations: pruned reading, dummy reading and encrypted reading.

Pruned reading provides only the reading of those parts of the XML document that are allowed to be read by the user – the content for which the user has no read authorization is removed. Given the extended DOM model of a document, this process is performed by removing nodes of the document tree for which the access is not granted. If a node, for which the access is not granted, is a leaf, it is removed from the tree. If a node with a denied access is not a leaf (hence it is an XML element) then, in order to preserve the document structure, it is not removed from the tree, but its attributes which the user is not granted to read are removed.

The **dummy read** operation processes the document in such a way that the parts of the document not granted to be read are replaced by dummy (fake) values. The purpose of this type of reading operation is to obtain the document which is in accordance with its XML schema. Since the XML Schema standard supports a large number of data types and defining new ones, generating dummy values according to the given data type may be very complex. We have opted to use the approach that multiple dummy value generators may be implemented and integrated into the XXACF, each targeting a specific data type.

The pruned reading and dummy reading operations are executed on user demand, i.e. on each request for reading a document. In case of a large number of users accesses the documents, mostly for reading, using previously described reading operations that can seriously impact the system performance.

The effective alternative to these approaches is to use **encrypted reading** – by creating a new document based on the original one, according to the access control policies defined for the original document. The users can access only the parts of the new document for which they are authorized. One of the methods to form that kind of document is to use cryptographic techniques based on keys (Stallings, 1998; Schneier, 1996). According to access control policies, different parts of document are encrypted with different keys. The user possesses only those keys that enable him or her to decrypt the parts of the documents that he or she is allowed to access. The major problem for this type of reading is to determine which document parts will be encrypted by which key. The simplest approach is to encrypt each document node with a different key, while this key is accessible only to the users authorized for the access to the node. This approach is simple for implementation, but can cause generation of large number of keys. Our solution to this problem is to determine role groups, where each group is consisted of all roles to which the access to some node(s) is granted. One key is generated for each role group; all nodes for which that group has the access right are encrypted by that key.

Activity diagram in Figure 6 describes the XXACF procedure of determining the list of the root nodes of the subtrees which each node should be encrypted by the same key, as well as the possible subtrees transformation. All nodes of the each subtree own same role group. If the node is an XML element, the method is recursively invoked for its attributes and child nodes. If the node is not an XML element, it is inserted in the list. In case that the node is an element, and that it has no attributes or child nodes (the element is a leaf node), it is inserted in the list. If the element node is not a leaf, it is verified if all its attributes and child nodes are in the list and whether they all have the same role group as the element node. If all these conditions are met, the whole subtree having the given element as a root can be encrypted by the same key. Therefore, all attributes and child nodes are removed from list, and the node is inserted. If one of these conditions is not satisfied, the transformations of attributes, as well as all child nodes that are not elements, are performed in order to enable the encryption with different keys. The XML Encryption specification allows only the encryption on the element level and it is possible to encrypt the whole element or its content only (XML Encryption, 2002). If an attribute's role group differs from the role group of its element parent, it is necessary to encrypt that attribute with another key. In order to enable attribute encryption and maintain conformance with XML Encryption specification, it is necessary to transform it into the element. The similar case occurs if it is necessary to encrypt the element content (child node) with different keys. Since it is not possible to encrypt the whole element with one key, it is necessary to transform all attributes to subelements of the given element. For the same reason, there are the situations when all not-element subnodes must be transformed. At the end of this activity encryption list will contain the root nodes of the subtrees which nodes will be encrypted by the one key. Also, all root nodes (subtreees) with the same role group should be encrypted by the same key.

5 CONCLUSIONS

This paper presents the main features of the eXtensible XML Role-Based Access Control Framework (XXACF). The language for access control definition provides its representation according to the RBAC model and enables definition of context-sensitive access control. The system supports specifying access control policies on document schema, document instance, and document fragment levels. Also, content-dependent access control policies specification is possible. XXACF provides access control enforcement for different operations on a document, as well as the possibility of different ways of access control enforcement for the same operation. XXACF is a Java-based application. The system extensibility and configurability facilitates the customization of a specific implementation to users' needs, on the basis of document access control. The software architecture, presented in this paper, separates XML documents from RBAC components and provides an independent design and administration of the access control policies.

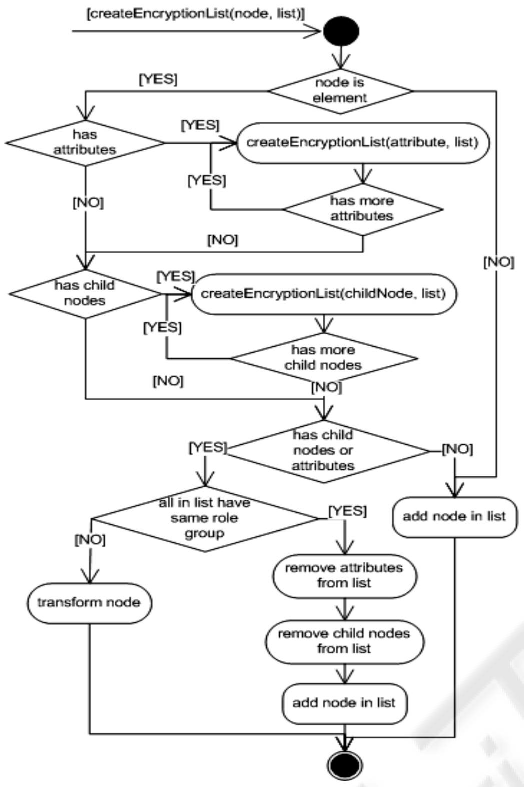
[createEncryptionList(node, list)]
node is element
[YES]
has attributes
[YES]
createEncryptionList(attribute, list)
[YES]
has more attributes
[NO]
[NO]
[NO]
has child nodes
[YES]
createEncryptionList(childNode, list)
[YES]
has more child nodes
[NO]
[NO]
has child nodes or attributes
[YES]
[NO]
all in list have same role group
[NO]
[YES]
add node in list
transform node
remove attributes from list
remove child nodes from list
add node in list

Figure 6: Creation of encryption node list in XXACF.

The most notable improvements over systems reviewed in Section 2 are the following:

- Access control is based on the RBAC model with support of the users' roles hierarchy.
- Definition of access control policies on different priority and granularity levels which may be content depended.
- Context-sensitive access control enforcement is supported.
- Support for separate access control enforcement for different operations on documents and different ways of implementing of the same operation.
- XXACF is extensible in the way that its deployment can be customized according to users' specific needs.

The XXACF prototype implementation is verified on a real workflow system based on XML documents.

Future work in development of XXACF includes the integration with other access control systems, enabling the application that uses XXACF operate not only with XML documents, but also with data in other formats (such as relational databases). Besides, the functionality of defining the static and dynamic separation of duties (SoD) constraints (Ferraiolo et. al., 2003) is also under way.

REFERENCES

Ferraiolo, D. F., Kuhn D. R., Chandramouli, R., 2003. *Role-Based Access Control*, Artech House.

Ferraiolo, D.F., Sandhu, R., Gavrila, S., Kuhn, D.R., Chandramouli, R., 2001. Proposed NIST Standard for Role-Based Access Control. *ACM Transactions Information and System Security*, 4(3), pp. 224-274.

Bhatti, R., Joshi, J.B.D., Bertino, E., Ghafoor, A., 2004. XML Based Specification for Web Services Document Security [Electronic version]. *IEEE Computer Society Press*, 37(4), pp. 41-49.

Bhatti, R., Joshi, J.B., Bertino, E., Ghafoo, A., 2003. Access Control in Dynamic XML-Based Web-Services with X-RBAC [Electronic version]. *The First International Conference on Web Services*, Las Vegas, USA.

Botha, R.A., Eloff, J.H.P., 2001. An Access Control Architecture for XML Documents in Workflow Environments [Electronic version]. *Proceedings of SAICSIT 2001*, *South African Computer Journal,* 3.

Botha, R.A., Eloff, J.H.P., 2001. A Framework for Access Control in Workflow Systems. *Information Management and Computer Security*, 9(3), pp 126–133.

Damiani, E., De Capitani Di Vimercati, S., Paraboschi, S., Samarati, P., 2000. Securing XML documents [Electronic version]. *Proceedings of the 7th International Conference on Extending Database Technology,* Konstanz, Germany, pp. 121-135.

Damiani, E., de Capitani di Vimercati, S., Paraboschi, S., Samarati, P., 2002. A Fine Grained Access Control System for XML Documents. *ACM Transactions on Information and System Security*, 5(2), pp. 169–202.

Bertino, E., Castano, S., Ferrari, E., 2001. Securing XML Documents with Author-X. *IEEE Internet Computing* 5(3), pp. 21–31.

Bertino, E., Ferrari, E., 2002. Secure and Selective Dissemination of XML Documents. *ACM Transactions Information and System Security*, 5(3). pp. 290–331.

Hada, S., Kudo, M., 2002. XML Access Control Language. Retrieved January 10. 2006, from http://www.trl.ibm.com/projects/xml/xss4j/docs/xacl-spec.html.

Stallings, W., 1998. *Cryptography and Network Security: Principles and Practice*, Prentice Hall.

Schneier, B., 1996. *Applied Cryptography*, John Wiley.

XML Encryption Syntax and Processing. Retrieved June 10. 2005, from http://www.w3.org/TR/xmlenc-core.