DOI: 10.2298/CSIS100827002S

Access Control Framework for XML Document Collections

Goran Sladić, Branko Milosavljević, Zora Konjović, and Milan Vidaković

Faculty of Technical Sciences, University of Novi Sad, Trg D. Obradovića 6
21000 Novi Sad, Serbia
{sladicg, mbranko, ftn_zora, minja}@uns.ac.rs

Abstract. It is often the case that XML documents contain information of different sensitivity degrees that must be selectively shared by user communities. 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 model of XXACF is described. The framework represents an improvement over the existing systems and enables defining context-sensitive access control policies on different priority and granularity levels, the enforcement of access control for different operations on XML documents, as well as different ways of access control enforcement for the same operation.

Keywords: access control, RBAC, context, XML.

1. Introduction

Access control is only one aspect of a comprehensive computer security solution, but also the 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 a role of a user in the system [21]. The 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 [21].

The growth of use of XML as a format for data modelling and interchange accentuates the issue of access control to XML documents. An XML document may contain data with different levels of accessibility. eXtensible XML Role-Based Access Control Framework (XXACF) [39] provides the means of defining access control policies and access control enforcement based on the RBAC model. Access control policies in XXACF may be defined on different priority and granularity levels and they may be content dependent, 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 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.

The rest of the paper is structured as follows. Section 2 reviews the related work. Section 3 presents the XXACF data model. The extended DOM model is presented in Section 4. Section 5 describes the procedure of access control enforcement. Section 6 concludes the paper and outlines further research directions.

2. Related Work

In [29] the authors present a provisional authorization model that provides XML with element-wise access control mechanism. They have formalized a provisional authorization model that adds extended semantics to traditional authorization models. The proposed model integrates several security features such as authorization and non-repudiation in unified XML documents and enables the authorization initiator not only to securely browse XML documents but also to securely update each document element.

In the paper [11] 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 [11]: 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* [11]. Each process in the workflow system implies manipulation of a 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 [11].

In the paper [25] the author defines a security model for a native XML database which supports XUpdate language. Since the model is implemented in a native XML database supporting XUpdate, a privilege that each XUpdate operation requires for completion needs to be specified. Presented mode is inspired by the SQL security model.

X-RBAC [9] [10] 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. Access control policies depend on the user's session context and the content of the documents being accessed [9]. Depending on the document content, the access control policies can be specified on four granularity levels: the conceptual level, the document schema level, the document instance level and the document element level [9]. However, policies that deny access cannot be specified. The supported operations are *reading*, *writing*, *navigating* (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 [10].

XML ACP (Access Control Processor) [16] [17] 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 [16]. The access control is

possible on several levels of granularity: the DTD level, the document instance level, the specific document element/attribute level. The access control policies can be classified into eight priority groups. Propagation of the access control policies down through the hierarchy is supported. The policies can grant or deny access [16] [17]. 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 [16] [17].

The Author-X system [6] [8] provides for defining access control policies on different granularity levels, which can grant or deny access. The controlled propagation of access control policies is provided, where policies defined for a document or DTD can be applied on other semantically related documents or DTDs on different granularity levels [8]. There are two working modes: *pull* and *push* [6]. In the pull mode, the user explicitly requests access to a document. After reception of the request, Author-X 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 if the changed content is in accordance with security rights [6]. 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 keys which are available to her or him according to the specified security rights [6].

Qi et al. [35] propose an approach to XML access control through rule functions that are managed separately from documents. The key idea is to encode the access control rules as a set of rule functions that separately perform the actual access evaluation. A rule function is an executable code fragment that encapsulates the access rules and is shared by all documents of the same type. According to authors, the novelties of this model are the high scalability and high performance.

Fundulaki and Maneth [24] propose language for specifying access control on XML data in the presence of update operations. The update operations used in this model are based on the W3C XQuery Update Facility specification. Alternative language that supports access control annotations at the level of the XML DTD is also presented.

In [36] access to a document and its parts can be defined based both on the current document content and on the history information that captures the operations performed on that document. Moreover, the history information includes the source of the parts of a document that were transferred from different documents.

Byun and Park [12] propose two phase filtering scheme for access control enforcement mechanism. The first phase filtering is to abstract only necessary access control rules based on a user query. The second phase filtering is to modify an unsafe query into a safe one. Query modification is the development

of an efficient query rewriting mechanism that transforms an unsafe query into a safe yet correct one that keeps the user access control policies.

An and Park [3] present access control labeling scheme for efficient secure query processing under dynamic XML data streams.

Knowledge based formal approach to ensure the security of web-based XML documents is presented in [4]. Given approach is based on a high level language to specify an XML document and its protection authorizations.

The focus of the paper [15] is how to control access to XML documents, once they have been received. The paper describes how certain access control policies for restricting access to XML documents can be enforced by encrypting specified regions of the document. The regions are specified using XPath filters and the policies based on the hierarchical structure of XML document. Objects are ordered using containment of their respective XPath expressions. They are encrypted with a number of different keys based on the relative seniority of objects. A user is supplied with a master key enabling her/him to decrypt those objects for which she/he is authorized.

Miklau and Suciu [31] propose framework for enforcing access control policies on published XML documents using cryptography. In this framework the owner publishes a single data instance, which is partially encrypted and which enforces all access control policies. The owner enforces an access control policy by granting keys to users. A client can access the data conditionally, depending on the keys she/he possesses. The client does not need to decrypt the entire data instance; it can access to data selectively using a query language. Authors also present declarative language for access control policies and extension to XQuery to support selective access to a document.

XACML (eXtensible Access Control Markup Language) [32] provides a general purpose access control language. It describes both an access control policy language and a request/response language. The policy language is used to express access control policies (who can do what when). The request/response language expresses queries about whether a particular access should be allowed (requests) and describes answers to those queries (responses). XACML also suggests a policy authorization model to guide implementers of the authorization mechanism. The hierarchical resource profile of XACML [33] specifies how XACML can provide access control for resources (including XML documents) that are organized as a hierarchy. The core and hierarchical role-based access control (RBAC) profile of XACML [34] defines a profile for the use of XACML to meet the requirements of RBAC. Haidar et al. [1] analyze RBAC profile of XACML and they identify several limitations of proposed profile.

Various definitions of context have been proposed in the literature [2] [13] [19] [23] [37]. Broadly, the notion of a context relates to the characterization of environment conditions that are relevant for performing appropriate actions in the computing domain. In order to realize the fine grained access control many context-sensitive access control models have been proposed. Many authors [20] [27] [38] [41] propose a context-based access control model for web services. Their approaches grant or adapt permissions to users based on a set

of contextual information collected from environments of the system. Covington et al. [14] introduce the notion of environmental role, and provide a uniform access control framework that can be used to secure context-sensitive applications. In the paper [22] authors showed the use of context information and its quality indicators to grant access permissions to resources. Georgiadis et al. [26] discuss the integration of contextual information with team and role-based access control. The influence of temporal constraints to access control is probably the most thoroughly analyzed in [5] [30], while the influence of geospatial constraints is presented in [7] [18].

By analyzing the previously mentioned XML access control frameworks, we notice that a significant number of them do not support the RBAC-based access control, but are based on different access control models. Although the literature recognizes a significant number of context-dependent access control models, the impact of context to XML access control is only partial. All presented XML access control frameworks support the access control enforcement for reading documents, but not for creating, updating and deleting documents. To the best of our knowledge, research on access control to XML document in the presence of XML Schema is very poor. The presented models usually implement the reading operation by pruning parts of documents for which users are not authorized. This approach may cause that the newly formed document is not in accordance with its schema. Cases when a user requests that the documents that she/he receives are in accordance with theirs XML Schema are only partially considered.

XXACF, the system presented in this paper, has the following notable improvements over the aforementioned XML access control systems:

- Access control is based on the RBAC model with the support for the role hierarchy
- Context-sensitive access control enforcement is supported
- Definition of granting and denying access control policies on different priority and granularity levels and document content-dependent policies are supported
- Support for separate access control enforcement for different operations on documents
- Different ways of implementing read operation on documents in order to provide: (a) pruned reading, (b) reading when t is required that a document is in accordance with its schema after access control enforcement and (c) specific reading to improve performance in case of a large number of accesses to a document

3. The XXACF Access Control Policies Model

The XML Schema in Figure 1 defines the structure of access control policies, which is based on policies described in [29] supporting policy conditions. Access control policies described in [29] are extended in order to support role hierarchies, to provide priorities of access control policies as defined in [16] [17]

and to support arbitrary operations for which policies are defined. We use our schema for brevity, although this model can be represented through the XACML language.

Each access control policy is defined by the `<policy>` element to which the unique identifier (the `id` attribute) is assigned. The policy (`<permission>`) permits or denies the subject to execute the operation (`<operation>`) on the object (`<object>`). 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. Policies defined for a document schema are applied to all document instances of that type (all documents that are valid according to the schema), while policies defined for a particular document instance are applied only to that instance. 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, an object (policy) which is content dependent, may be specified.

XXACF can support context-sensitive access control and dynamic capturing of context information through `<condition>` element. If the condition is satisfied, the access control policy will be applied. Otherwise, it will not be the case. Each condition consists of the name of a logical operation (*not*, *and*, *nand*, *or*, *nor*, and *xor*) (attribute `operation`) and subconditions and/or predicates (`<predicate>` element). A predicate is a function which returns a Boolean value. The value calculated by applying a specified logical operation on return values of subconditions and predicates represents the return value of the condition. Functionality of a predicate should be properly implemented. This facilitates the implementation of the specific access control which is not supported through the RBAC model. For example, insertion of a digital signature into a document is allowed only if the signature is valid and is generated using the valid key of the logged user. This verification can be realized by implementing a predicate that will verify a signature of the document using the certificate of the logged user and also verify if the certificate is revoked. By using a predicate where the return value depends on the state of the environment (e.g. access to the document depends on the current state of the process which the document belongs to), it is possible to implement context-sensitive access control.

Although the standard RBAC model supports only granting policies, results in the literature (see Section 2) identify a need for denying policies to achieve more efficient security administration. The permission is described by the element `<permission>` shown in Figure 1. The permission can be dual, granting access (the value of the `type` attribute is "*grant*") or denying it (the value of the `type` attribute is "*deny*"). In order to avoid specifying explicitly the policy for each entity, propagation of policies is enabled, starting from the entity specified by the `<object>` element down or up the hierarchy. The `propagation_direction` attribute 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

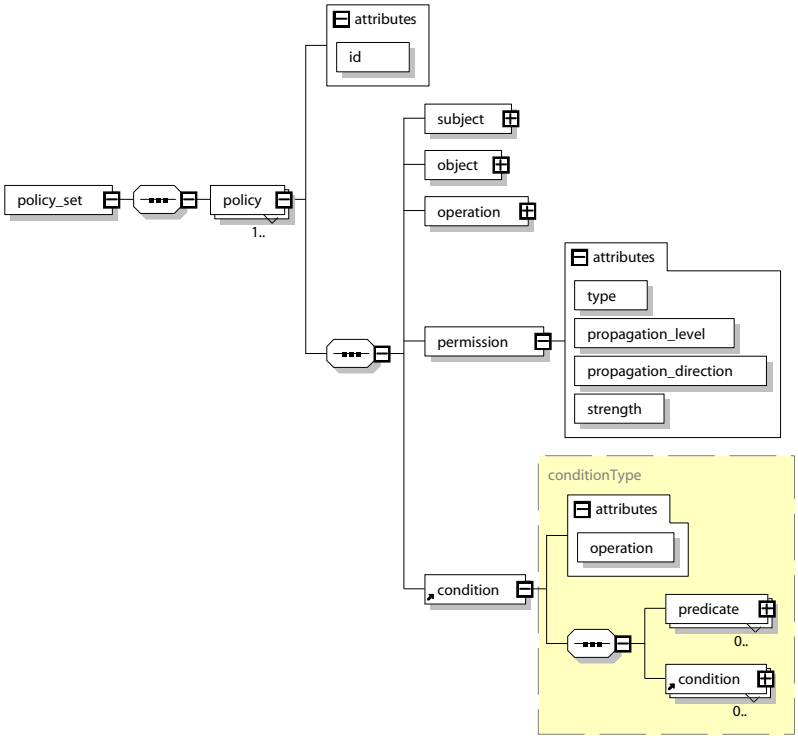

Fig. 1. XML Schema of access control policies

object upwards. The level of propagation, i.e. the maximum number of hierarchy levels where the propagation is performed, is specified by the attribute `propagation_level`. The propagation level can be arbitrary, it can include a certain number of levels, or with no propagation at all. Access control policy strength is defined by the `strength` attribute as defined in [17]. 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 a document schema or a document instance, "*hard*" is specified only for a document schema and "*soft*" if the object is a document instance. Detailed explanation of the semantics of the `strength` attribute and it's values can be found in [17].

Access control policies defined for a descendant role have a higher priority than the policies defined for an ascendant role. The priority of access control policies defined for the same role is determined according to the object they are related to (a document schema or a 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 presented in Table 1. We believe that this approach for determining priorities is more applicable in practice than the explicit priority assignment, because the explicit assignment does not necessary have the semantics of the priority values.

Table 1. Priority levels of access control policies

Priority level	Schema/Instance	Propagation	Strength
1	schema	no	hard
2	schema	yes	hard
3	instance	no	normal
4	instance	yes	normal
5	schema	no	normal
6	schema	yes	normal
7	instance	no	soft
8	instance	yes	soft

4. The Extended DOM Model

The process of access control enforcement for XML documents in XXACF is performed on DOM (Document Object Model) of the document. In order to achieve access control, the DOM model is extended with necessary functionality.

The diagram in Figure 2 shows the classes which represent a node in a document tree. Each implementation of the `XACNode` interface contains the `MarkMap` object. It contains all access control policies applied on that node (Figure 2 shows the example for `XACElementImpl`). The role for which the access control policy is defined is used as a key to the hash map contained in the `MarkMap` class. Hash map values are instances of the `MarkMatrix` class. This class contains access control policies defined for the given role and applied to that node.

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

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

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 to these nodes.

To avoid implementation of the functionality defined by the interfaces of the DOM model (interfaces from the package `org.w3c.dom`), *Apache Xerces* implementation of the DOM model is used. An example of implementation of the `XACElement` interface is shown in Figure 2. The class `XACElementImpl` represents implementation of `XACElement`. `ElementImpl` is Xerces implementation of the `Element` interface. By extending this class, implementation of the functionality defined by `Node` and `Element` interfaces is avoided. Other specializations of the `XACNode` interfaces are implemented in the same way.

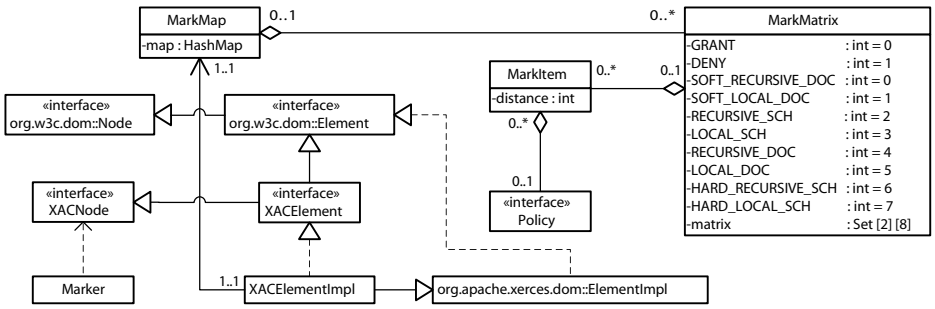

Fig. 2. Classes for marking a node

5. Access Control Enforcement

The process of access control is performed in four steps:

1. selection of the applicable access control policies,
2. marking document nodes,
3. conflict resolution, and
4. execution of the requested operation.

These four steps are described in the following subsections.

The description of the access control enforcement process adopts the following notions.

The set of all document schemas (S) is comprised of all document schemas defined in a system, i.e. $S = \{St_i \mid 0 < i \leq n\}$ where St_i is the i-th document schema and n is the number of schemas in the system. Each St_i consists of the main XML Schema and all XML Schemas that are included in the main.

Let D be the set of all documents in the system and Dt_i be the set of all documents of a particular type t_i (all documents which are in accordance with a certain schema). Then D is defined as follows $D = \bigcup_{i=1}^{n} Dt_i$, where n is the number of different schemas (document types) in the system.

The set of all access control policies is defined as follows $P = PD \cup PS$, where: $PD = \bigcup_{i=1}^{n} PDt_i, PS = \bigcup_{i=1}^{n} PSt_i$, PD is the set of all policies defined for documents, PDt_i is the set of all policies defined for documents belonging to the set Dt_i, PS is the set of all policies defined for document schemas and PSt_i is the set of policies defined for the document schema St_i.

Let R be the set of all user roles, U is the set of all users in the system and O is the set of operations defined in the system.

$markItem$ is defined as a tuple of the form: $markItem = (policy, distance)$, where: $policy \in PDt_i \cup PSt_i$ and $distance \in N_0$ is the distance between the root node selected by the object of the policy and the node to which the access control policy is applied.

5.1. The Selection of the Applicable Access Control Policies

The goal of this task is to find access control policies that will be used for access control enforcement on the document being accessed.

When a document $doc \in Dt_i$ is being accessed in order to execute a certain operation, only access control policies that apply to the document doc from PDt_i and its schema (PSt_i) 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 (if the condition is specified) for each loaded access control policy. If the condition is satisfied, the policy will be applied (it is added to applicable policies set, APS); otherwise it will not be applied.

5.2. Marking Document Nodes

Marking of document nodes is a process of determining and applying each policy from the applicable policy set to the nodes selected by that policy.

Depending on if the propagation direction for a policy is down ("*DOWN*") or up ("*UP*") the hierarchy (see Section 3), the appropriate propagation is performed until the maximum level of propagation is reached. A `MarkItem` instance is created for each selected node and added to the appropriate instance of `MarkMatrix` of the node's `MarkMap`.

Upon completing this task, each node will be associated with access control policies to be applied to it.

5.3. Conflict Resolution

Since XXACF supports granting and denying policies it is possible that on some nodes both policy types are applied. In these cases a conflict need to be resolved, i.e. determine whether granting or denying policies will be applied. Our procedure for conflict resolving is presented in Algorithm 1. The $NodeConflictResolve$ function is invoked for each node of the specified subtree to resolve the conflict for the given node.

The first activity of selecting the final access control policies for the given node is conflict resolution according to the "*more specific subject takes precedence*" (MSSTP) principle (invoking the $ResolveConflictByMSSTPPrinciple$ function). According to this principle, access control policies defined for more specific roles have a greater priority than the ones defined for less specific roles (if the role r_i extends the role r_j then r_i has greater priority). For each user's role it is checked if there are policies defined for that role. If the policies defined for that role exist, they are selected. On the other hand, if they do not exist, the role hierarchy is recursively traversed in search of policies.

Next, the conflict resolution using the "*more specific object takes precedence*" (MSOTP) principle is performed on the policies previously retreived. In this step, the policies where the object is nearest to the node which the policy is applied to are selected. First, in the grant policies set (Grant Mark Item Set,

Algorithm 1 Conflict resolution for a node

```
NAME: NodeConflictResolve
INPUT: node - node for which conflict resolving is preformed
URS ⊆ R - user's roles
defaultConflictResolution - default conflict resolution for document
defaultPolicy - default policy
OUTPUT: node - node with determined final policies

Let FPS be the set of final policies for node
Let markMap be the mark map of node
markMatrix := CreateEmptyMarkMatrix()
markMatrix := ResolveConflictByMSSTPPrinciple(markMap, URS, markMatrix)
collIndex := FindTheHigestNonEmptyColumn(markMatrix)

{If there is such a column, resolve conflict for policies in that column using
the MSOTP principle}
if collIndex ≠ −1 then
  Let GMIS be the mark item set with grant policies in markMatrix at the
  column collIndex
  GMIS := KeepPoliciesWithMSO(GMIS)
  Let DMIS be the mark item set with deny policies in markMatrix at the
  column collIndex
  DMIS := KeepPoliciesWithMSO(DMIS)
  MIS := ResolveConflictByMSOTPPrinciple(GMIS, DMIS)

  if HasGrantPolicies(MIS) ∧ HasDenyPolicies(MIS) then
    if defaultConflictResolution = grant_takes_precedence then
      FPS := GetGrantPolicies(MIS);
    else
      FPS := GetDenyPolicies(MIS);
    end if
  else if HasGrantPolicies(MIS) then
    FPS := GetGrantPolicies(MIS);
  else if HasDenyPolicies(MIS) then
    FPS := GetDenyPolicies(MIS);
  else
    FPS := {defaultPolicy} {No policies}
  end if
else
  FPS := {defaultPolicy} {No policies}
end if
SetFinalPolices(node, FPS)
```

$GMIS$) only policies with the minimal distance are kept and the remaimning policies are removed. The same action is preformed on the denial policies set (Deny Mark Item Set, $DMIS$). The result of conflict resolution using the MSOTP principle is the one of these two sets ($GMIS$ or $DMIS$) which has the minimal distance, or the union of these two sets ($GMIS \cup DMIS$) if the distance is the same.

At the end of the conflict resolution process it is checked if there are access control policies that grant and deny access. If there are both, the default conflict resolution ($defaultConflictResolution$) is verified. In the case that in the selected priority level all access control policies either grant or deny access, the final access control policies either grant access, or deny it. If no policies are applied to a node, then depending on the value of the *default policy*, access to that node will be granted or denied.

After the conflict resolutiont each node is associated with its final access control policies, either granting or denying access. There may be more than one final policy associated with a node.

5.4. Executing Operations in XXACF

XXACF supports the following operations on XML documents:

- updating documents (**adding** new nodes, **deleting** and **changing** (replacing) the existing ones) and
- reading documents. The reading operation is implemented in three different ways: **pruned reading**, **fake reading** and **encrypted reading**.

Modifying Operations The complete process of adding new nodes into a tree along with access control enforcement is defined by Algorithm 2. Each new subtree to be *inserted* into the document tree is temporarily inserted. Then, the selected access control policies are applied and the conflict resolution process on the *inserted nodes* is performed. If the final access control policies on each node of the inserted subtree grant access, the insertion is allowed. If insertion of any node of the inserted subtree is not allowed, the subtree is removed from the document tree.

Algorithm 2 Add operation

NAME: AddOperation
INPUT: $session$ - user's session
$doc \in Dt_i$ - document being accessed
$ANS = \{(node, pos) \mid node$ - root node of subtree, pos - position in document where node should be inserted$\}$ - set of subtree root nodes which are added to the document
$URS \subseteq R$ - user's roles
OUTPUT: doc - document with new nodes if insertion of new nodes is allowed

{Conditialnally insert all new nodes in the document on the specified position}
for each $item \in ANS$ **do**
  doc_i := InsertNodeInDocument(doc_i, $item.node$, $item.pos$)
end for

sch_i := GetDocumentSchema(doc_i)
APS := GetApplicablePolicies($session$, doc_i, sch_i, add, USR)
$xdom$:= MarkDocument(doc_i, APS)

for each $item \in ANS$ **do**
  $xdom$:= ResolveConflict($xdom$, $item.node$, URS)
  if ¬InsertionOfSubtreeAllowed($xdoc$, $item.node$) **then**
    $xdoc$:= RemoveSubtree($xdoc$, $item.node$)
  end if
end for
doc := Transform($xdom$)

In the same way as in the case of the insertion operation, the **delete** operation of the subtree is allowed if deletion of all nodes of the subtree is allowed,

i.e. if the final access control policies of each node of the subtree permit this operation.

For **replacement** of a 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 (if the node can be replaced). The second set defines the postcondition (if replacement with the given node is allowed).

Reading Operations XXACF currently supports three types of reading operations: pruned reading, fake reading and encrypted reading.

Pruned reading provides only 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 access is not granted. If a node, for which access is not granted, is a leaf, it is removed from the tree. If a node with 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 **fake reading** operation processes the document in such a way that the parts of the document not granted to be read are replaced by fake (dummy) 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 the large number of data types and defining new ones, generating fake values according to the given data type may be very complex. We have opted for using the approach that multiple dummy value generators may be implemented and integrated into XXACF, each targeting a specific data type.

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

An 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. 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. According to access control policies, different parts of a document are encrypted with different keys. A user possesses only those keys that enable him or her to decrypt the parts of the documents that she or he 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 users authorized for access to that node. This approach is simple for implementation,

but can cause generation of a large number of keys. Our solution to this problem is to determine *role groups*, where each group consists of all roles to which 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.

The XML Encryption specification allows only encryption on the element level and it is possible to encrypt the whole element or its content only [28]. If an attribute role group differs from the role group of its element, it is necessary to encrypt that attribute with another key. In order to enable attribute encryption and maintain conformance with the XML Encryption specification, it is necessary to transform it into the element. The similar case occurs if it is necessary to encrypt an element content (child nodes) 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 situations when all non-element subnodes must be transformed.

Algorithm 3 Creation of an encrypted node set

```
NAME: CreateEncryptionDOM
INPUT: node - currently processed node
ENS - contains root nodes of subtree which are encrypted with the same key in
the previous call
OUTPUT: node - modifyed node, if required in order to enable encryption using
XML Encryption
ENS - set which contains root nodes of subtree which are encrypted with same
key

Let ANS be attributes set of the node
Let CNS be child nodes set of the node
if ANS ≠ ∅ ∨ CNS = ∅ then
  if ANS ≠ ∅ then
    for each attr ∈ ANS do
      CreateEncryptionDOM(attr, ENS) {Recursive call}
    end for
  end if

  if CNS ≠ ∅ then
    for each childNode ∈ CNS do
      CreateEncryptionDOM(childNode, ENS) {Recursive call}
    end for
  end if

  if SameRoleGroup(node, ANS, CNS) then
    RemoveNodes(ANS, ENS)
    RemoveNodes(CNS, ENS)
    AddNode(node, ENS)
  else
    node := TransformNode(node, ENS)
  end if
else
  AddNode(node, ENS)
end if
```

Determining role groups is defined by Algorithm 3. It describes the our procedure for both determining the set of the root nodes of the subtrees in which each node should be encrypted by the same key, and possible subtrees transformation.

Since, the secret keys are stored into the document it is necessary to encrypt those keys with the public keys of users who will read the document using the asymmetric encryption procedure. Only secret keys assigned to user's roles are encrypted with the public key of each user (i.e., keys that belong to role groups which contain one or more user's roles). Then a user will be able to decrypt only secret keys which are encrypted with her/his public key. In this manner, a user can access only secret keys assigned to her/his roles and therefore she/he can decrypt only document fragments which she/he is authorized for.

6. Conclusion

This paper presents the main features of eXtensible XML Role-Based Access Control Framework (XXACF).The proposed access control model provides access control representation according to the RBAC model and enables definition of the context-sensitive access control. It allows specification of the access control policies on the document schema, instance, and 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. The presented model separates XML documents from RBAC components and provides independent design and administration of access control policies.

The most notable improvements over the systems reviewed in Section 2 include: (a) context-sensitive access control based on the hierarchical RBAC model, (b) document-dependent definition of access control policies on different priority and granularity levels, and (c) support for separate access control enforcement for different operations on documents and different ways of implementing the same operation.

The XXACF prototype implementation is verified on a document-centric workflow system based on XML documents. The presented prototype implementation represents the proof of the proposed model practical value. Response time for the most documents was satisfactory. The time for access control enforcement is significantly less than the overall time to a document access. Only for a few quite large documents, the access control enforcement time is a significant part of the total access time. XXACF is yet to be verified in different environments. So far we have not considered performance implications outside of the prototype workflow system.

Future work in XXACF development includes the integration with other access control systems, enabling an application that uses XXACF to operate not only with XML documents, but also with data in other formats (such as relational databases). Moreover, we plan to adjust XXACF for access control in distributed agent-based systems like [40]. We think that a formal specification of the context has to be done in order to enable efficient usage of XXACF in different

systems. Functionality of defining constraints of static and dynamic separation of duties (SoD) is also under way.

References

1. Abi Haidar, D., Cuppens-Boulahia, N., Cuppens, F., Debar, H.: An extended RBAC profile of XACML. In: SWS '06: Proceedings of the 3rd ACM workshop on Secure web services. pp. 13–22. ACM, New York, NY, USA (2006)
2. Abowd, G.D., Dey, A.K., Brown, P.J., Davies, N., Smith, M., Steggles, P.: Towards a better understanding of context and context-awareness. In: HUC '99: Proceedings of the 1st international symposium on Handheld and Ubiquitous Computing. pp. 304–307. Springer-Verlag, London, UK (1999)
3. An, D.C., Park, S.: Access control labeling scheme for efficient secure XML query processing. In: Proceedings of the 12th international conference on Knowledge-Based Intelligent Information and Engineering Systems, Part II. pp. 346–353. Springer, Berlin (2008)
4. Bai, Y.: Access control for XML document. In: IEA/AIE '08: Proceedings of the 21st international conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems. pp. 621–630. Springer-Verlag, Berlin, Heidelberg (2008)
5. Bertino, E., Bonatti, P.A., Ferrari, E.: TRBAC: A temporal role-based access control model. ACM Trans. Inf. Syst. Secur. 4(3), 191–233 (2001)
6. Bertino, E., Castano, S., Ferrari, E.: Securing XML documents with Author-X. IEEE Internet Computing 05(3), 21–31 (2001)
7. Bertino, E., Catania, B., Damiani, M.L., Perlasca, P.: GEO-RBAC: a spatially aware RBAC. In: SACMAT '05: Proceedings of the tenth ACM symposium on Access control models and technologies. pp. 29–37. ACM, New York, NY, USA (2005)
8. Bertino, E., Ferrari, E.: Secure and selective dissemination of XML documents. ACM Trans. Inf. Syst. Secur. 5(3), 290–331 (2002)
9. Bhatti, R., Bertino, E., Ghafoor, A., Joshi, J.B.: XML-based specification for web services document security. Computer 37(4), 41–49 (2004)
10. Bhatti, R., Joshi, J.B., Bertino, E., Ghafoo, A.: Access control in dynamic XML-based web-services with X-RBAC. In: 1st International Conference on Web Services (2003)
11. Botha, R.A., Eloff, J.H.: A framework for access control in workflow environments. Information Management and Computer Security 9(3), 126–133 (2001)
12. Byun, C., Park, S.: Two phase filtering for XML access control. In: Secure Data Management. pp. 115–130. Springer (2006)
13. Chen, G., Kotz, D.: A survey of context-aware mobile computing research. Tech. rep., Hanover, NH, USA (2000)
14. Covington, M.J., Long, W., Srinivasan, S., Dev, A.K., Ahamad, M., Abowd, G.D.: Securing context-aware applications using environment roles. In: Proceedings of the 6th ACM Symposium on Access Control Models and Technologies (SACMAT). pp. 10–20. ACM, New York, NY, USA (2001)
15. Crampton, J.: Applying hierarchical and role-based access control to XML documents. In: SWS '04: Proceedings of the 2004 workshop on Secure web service. pp. 37–46. ACM, New York, NY, USA (2004)
16. Damiani, E., Samarati, P., di Vimercati, S.D.C., Paraboschi, S.: Controlling access to XML documents. IEEE Internet Computing 5(6), 18–28 (2001)

17. Damiani, E., di Vimercati, S.D.C., Paraboschi, S., Samarati, P.: A fine-grained access control system for XML documents. ACM Trans. Inf. Syst. Secur. 5(2), 169–202 (2002)
18. Damiani, M.L., Bertino, E., Catania, B., Perlasca, P.: GEO-RBAC: A spatially aware RBAC. ACM Trans. Inf. Syst. Secur. 10(1), 2 (2007)
19. Dey, A.K.: Understanding and using context. Personal Ubiquitous Comput. 5(1), 4–7 (2001)
20. Feng, X., Jun, X., Hao, H., Li, X.: Context-aware role-based access control model for web services. Grid and Cooperative Computing GCC 2004Workshops, International Workshop on Information Security and Survivability for Grid 3252, 430–436 (2004)
21. Ferraiolo, D.F., Sandhu, R., Gavrila, S., Kuhn, D.R., Chandramouli, R.: Proposed NIST standard for role-based access control. ACM Trans. Inf. Syst. Secur. 4(3), 224–274 (2001)
22. Filho, J.B., Martin, H.: Using context quality indicators for improving context-based access control in pervasive environments. In: EUC '08: Proceedings of the 2008 IEEE/IFIP International Conference on Embedded and Ubiquitous Computing. pp. 285–290. IEEE Computer Society, Washington, DC, USA (2008)
23. de Freitas Bulcao Neto, R., da Graca Campos Pimentel, M.: Toward a domain-independent semantic model for context-aware computing. In: Proceedings of the 3rd Latin American Web Congress (LA-WEB). pp. 61–70. IEEE Computer Society, Washington, DC, USA (2005)
24. Fundulaki, I., Maneth, S.: Formalizing XML access control for update operations. In: SACMAT '07: Proceedings of the 12th ACM symposium on Access control models and technologies. pp. 169–174. ACM, New York, NY, USA (2007)
25. Gabillon, A.: An authorization model for XML databases. In: SWS '04: Proceedings of the 2004 workshop on Secure web service. pp. 16–28. ACM, New York, NY, USA (2004)
26. Georgiadis, C.K., Mavridis, I., Pangalos, G., Thomas, R.K.: Flexible team-based access control using contexts. In: SACMAT '01: Proceedings of the sixth ACM symposium on Access control models and technologies. pp. 21–27. ACM, New York, USA (2001)
27. Haibo, S., Fan, H.: A context-aware role-based access control model for web services. Proceedings of the IEEE International Conference on e-Business Engineering (ICEBE) 0, 220–223 (2005)
28. Imamura, T., Dillaway, B., Simon, E.: XML encryption syntax and processing. W3C Recommendation (2002), http://www.w3.org/TR/xmlenc-core/
29. Kudo, M., Hada, S.: XML document security based on provisional authorization. In: CCS '00: Proceedings of the 7th ACM conference on Computer and communications security. pp. 87–96. ACM Press (2000)
30. Latif, U., Joshi, J.B.D., Bertino, E., Ghafoor, A.: A generalized temporal role-based access control model. IEEE Trans. on Knowl. and Data Eng. 17(1), 4–23 (2005)
31. Miklau, G., Suciu, D.: Controlling access to published data using cryptography. In: VLDB '2003: Proceedings of the 29th international conference on Very large data bases. pp. 898–909. VLDB Endowment (2003)
32. OASIS: Extensible access control markup language (XACML) v 2.0. OASIS Specification (2005), www.oasis-open.org/committees/xacml
33. OASIS: Hierarchical resource profile of XACML. OASIS Specification (2005), www.oasis-open.org/committees/xacml
34. OASIS: XACML profile for role based access control (RBAC). OASIS Specification (2005), www.oasis-open.org/committees/xacml

35. Qi, N., Kudo, M., Myllymaki, J., Pirahesh, H.: A function-based access control model for XML databases. In: CIKM '05: Proceedings of the 14th ACM international conference on Information and knowledge management. pp. 115–122. ACM, New York, NY, USA (2005)
36. Roder, P., Tafreschi, O., Eckert, C.: History-based access control for XML documents. In: ASIACCS '07: Proceedings of the 2nd ACM symposium on Information, computer and communications security. pp. 386–388. ACM, New York, NY, USA (2007)
37. Schilit, B., Adams, N., Want, R.: Context-aware computing applications. In: Proc of IEEE Workshop on Mobile Computing Systems and Applications. pp. 85–91. IEEE Computer Society, Washington, DC, USA (1994)
38. Shang, C., Yang, Z., Liu, Q., Zhao, C.: A context based dynamic access control model for web service. In: International Conference on Embedded and Ubiquitous Computing, IEEE/IFIP. vol. 2, pp. 339–343. IEEE Computer Society, Los Alamitos, CA, USA (2008)
39. Sladić, G., Milosavljević, B., Konjović, Z.: Extensible access control model for XML document collections. In: ICETE SECRYPT: Proceedings of the 2nd International Conference on Security and Cryptography. pp. 373–380. INSTICC (2007)
40. Vidaković, M., Milosavljević, B., Konjović, Z., Sladić, G.: Extensible java EE-based agent framework and its application on distributed library catalogues. Computer Science and Information Systems 6(2), 28 (2009)
41. Wolf, R., Keinz, T., Schneider, M.: A model for context-dependent access control for web-based services with role-based approach. Proceedings of the 14th IEEE International Workshop on Database and Expert Systems Applications (DEXA) 00, 209–214 (2003)

Goran Sladić is teaching assistant at the Faculty of Technical Sciences, Novi Sad, Serbia. Mr. Sladić received his Bachelor degree (2002) and Master degree (2006) all in Computer Science from the University of Novi Sad, Faculty of Technical Sciences. Since 2002 he is with the Faculty of Technical Science in Novi Sad. His research interests include information security, context-aware computing and document management. He is the corresponding author and can be contacted at: sladicg@uns.ac.rs

Branko Milosavljević is holding the associate professor position at the Faculty of Technical Sciences, Novi Sad, Serbia since 2008. Mr. Milosavljević received his Bachelor degree (1997), Master degree (1999), and PhD degree (2003) all in Computer Science from the University of Novi Sad, Faculty of Technical Sciences. Since 1998 he is with the Faculty of Technical Science in Novi Sad. His research interests include information retrieval, digital libraries, document management and information security.He can be contacted at: mbranko@uns.ac.rs

Zora Konjović is holding the full professor position at the Faculty of Technical Sciences, Novi Sad, Serbia since 2003. Mrs. Konjović received her Bachelor degree in Mathematics from the University of Novi Sad, Faculty Science in 1973, Master degree (1985) and Ph. D. degree (1992) booth in Robotics from the University of Novi Sad, Faculty of Technical Sciences. Since 1973 till 1980 she was

with the Faculty of Science in Novi Sad, and since 1980 she is with the Faculty of Technical Sciences, University of Novi Sad. Her research includes the fields of artificial intelligence, intelligent document management, and the applications of formal modeling in education. She can be contacted at: ftn_zora@uns.ac.rs

Milan Vidaković is holding the associate professor position at the Faculty of Technical Sciences, Novi Sad, Serbia. He received his PhD degree (2003) in Computer Science from the University of Novi Sad, Faculty of Technical Sciences. Since 1998 he has been with the Faculty of Technical Sciences in Novi Sad. Mr. Vidaković participated in several science projects. He published more than 60 scientific and professional papers. His main research interests include web and internet programming, distributed computing, software agents, and language internationalisation and localisation. He can be contacted at: minja@uns.ac.rs

Received: August 27, 2010; Accepted: April 5, 2011.