An RBAC-based PIB for Provisioning Access Control

Timothy E. Squair, Edgard Jamhour, Ricardo C. Nabhen

Pontifícia Universidade Católica do Paraná, PUCPR, PPGIA,
Rua Imaculada Conceição 1155, CEP 80215-901, Curitiba, Brazil.

{timothy, jamhour, rcnabhen}@ppgia.pucpr.br

Abstract. *This paper presents a framework for representing and distributing access control policies in distributed heterogeneous systems. Access control polices follows the RBAC (Role Based Access Control) model proposed by the NIST. The framework is based on the provisioning strategy defined by IETF, i.e., the RBAC information is represented in terms of a PIB (Policy Information Base) and distributed to the enforcement elements using the COPS-PR protocol. This approach can be explored in several scenarios for configuring both network devices and RBAC-aware applications. A research prototype has been implemented, and the results obtained from a performance analysis of the proposed extensions are summarized and evaluated.*

1. Introduction

Managing security in a large enterprise can be a complex task. Usually, applications developed using heterogeneous technologies adopt different strategies for representing and enforcing security. In this scenario, it is impractical to obtain a unified view of the security policies. Also, auditing and modifying security policies can be an expensive and difficult task. PBNM (Policy Based Network Management) is a promising approach for addressing this problem. A complete PBNM framework for managing security should include an unified model for representing policies, users and resources and also the mechanisms for distributing and enforcing these policies among heterogeneous applications.

This paper presents a framework for distributing access control policies in distributed heterogeneous systems. The framework is inspired on the recently published IETF standards concerning both policy representation and policy distribution adopting a provisioning approach. The provisioning approach is based on three main elements [2]:

i. a device-independent policy information model, used for representing policies that can be reused among different devices;

ii. a policy information base (PIB), which represents the policy assigned to a specific device. The PIB is generated from the device-independent policy model by a policy translation process. The translation takes into account the device capabilities, i.e., the mechanisms that the specific device supports for enforcing the policy;

iii. a protocol (COPS-PR) [3] specifically designed for supporting policy provisioning using the PIB structure, i.e., negotiating capabilities, transporting and installing the PIB into the device.

The provisioning approach is understandably generic, and can be explored in several management domains. IETF has already explored the provisioning approach for distributing *diffserv* configuration and, recently, for distributing *IPsec* configuration. Other potential target domains for future standardization are MPLS, Access Control and 3GPP UMTS [15]. Each management domain is addressed by defining a device-independent policy information model and a PIB. IETF has published the guidelines for defining these elements. The Policy Core Information Model (PCIM) [5] and its extensions (PCIMe) [6] define a generic set of classes and associations used for representing policies for any management domain. Policy models for specific domains are defined by extending the PCIM/PCIMe classes and associations. The framework PIB [9] defines a generic PIB template, which specifies the elements required for supporting capability negotiation and policy installation. Again, specific domains are addressed by extending the framework PIB elements.

This paper addresses a domain not yet explored by IETF, i.e., a framework for distributing RBAC (Role Based Access Control) policies to devices and applications. The framework is defined by introducing a device-independent RBAC information model and a RBAC-PIB.

The remaining of this paper is organized as follows: Section 2 discusses the motivation and contributions of this work. Section 3 reviews some important works that propose alternative approaches for access control policy representation and distribution. Section 4 describes the main elements of the proposed provisioning framework. Section 5 discusses the RBAC-based information model. Section 6 describes the RBAC-PIB. Section 7 presents the performance evaluation of the proposed framework, considering both: the size of the RBAC-PIB and the required time for provisioning the PIB. Finally, the conclusion summarizes the main aspects of this project and main points to future works.

2. Motivation and Contributions

The access control is one of the most important and complex aspects of the security management. The need of access control is present in the several components of a distributed system. In some cases, the access control refers to the right of managing network devices, such as gateways and firewalls. In other cases, the access control policies restrict the access of users to shared resources and application level services. RBAC is a quite generic information model that can be used for representing several types of access control policies. The RBAC NIST model, adopted in this work, offers a wide range of combinations where roles, permissions and resources can be associated in order to express a wide range of access control policies. Additionally, the RBAC NIST model allows to define special rules for constraining undesirable combinations of access permissions using static (session independent) and dynamic (activated in a session) restrictions.

In a distributed system, the access control is implemented by different elements. Not all elements can support the same enforcement mechanisms or are capable of interpreting all policy elements defined by the generic RBAC NIST model. The provisioning strategy offers an elegant approach for this problem. The strategy defines (at least) two information model levels: a device-independent information model and another that takes into account the capabilities of the device (PIB). In our work, these models have been called, respectively, RBPIM (Role-Based Policy Information Model) and RBAC-PIB (RBAC Policy Information Base).

The RBPIM allows to explore the RBAC model from a centralized point of view, allowing the reuse of elements shared by the several devices of the distributed environment (e.g. the role "Network Manager"). Complementarily, the translation process from RBPIM to RBAC-PIB allows to adapt the RBAC-policies according to the capabilities of the device by eliminating or replacing the elements not supported by the device.

The use of the COPS-PR protocol also brings evident advantages for distributing policies in a distributed and heterogeneous environment. COPS-PR is specifically designed for supporting policy provisioning using the PIB structure, i.e., negotiating capabilities, transporting and installing the PIB into the devices.

The major contribution of this paper is the RBAC-PIB definition. The RBPIM is based on a previous work Nabhen et al [12]. In this previous work, the RBPIM model have been explored using the outsourcing approach. Because some extensions have been included in order to support the provisioning approach, the RBPIM is also discussed in this paper. However, it is important to note that, because the PIB is defined by a policy translation, other information models representing RBAC policies could be combined with the PIB strategy. Section 3 will review some related works that can eventually be combined with the RBAC-PIB approach.

3. Other Aproaches

When defining an access control framework two important issues must be considered: (i) the model (or language) adopted for representing policies; (ii) the approach adopted for interpreting, distributing and enforcing the policies. The work described in this paper adopts a PCIM/PCIMe extension for representing RBAC policies and proposes a PIB/COPS-PR approach for distributing the policies. This work also adopts the PDP (Policy Decision Point) and PEP (Policy Enforcement Point) as the entities responsible, respectively, for policy interpretation/distribution and policy enforcement, as defined by IETF [2]. This section will review three other strategies for representing, distributing and enforcing access control policies.

An important work that adopts the PDP/PEP approach for access control is the XACML (eXtensible Access Control Markup Language), proposed by the OASIS consortium [10]. XACML is a complete solution for modeling, storing and distributing descriptive access control policies. The XACML adopts a generic access control model, based on the concept of policies, rules and targets. A XACML Target is a triple formed by subject, resource and action. Targets are used for selecting policies must be considered to evaluate a decision request and also for determining if a request is permitted or denied. By properly defined rules and policies, it is possible to model RBAC policies using the "*xacml policy*" language. In fact, the OASIS already published a document supplying the directives for using XACML for describing RBAC policies [11]. XACML-based frameworks are supposed to be implemented using an "outsourcing" PDP/PEP architecture, i.e., policy interpretation is performed by the PDP, and final decisions (Permit or Deny) are delivered to the PEPs. It is a logical assumption to consider that a "pure" outsourcing strategy represents a scalability issue when large scale systems are considered. Another important aspect refers to the "reuse" of management information. Policies are described in terms of subjects (e.g., users) and resources (e.g., applications). An important feature for an access control framework is the capacity of reusing management information shared with other management frameworks. In this scenario, describing policies using a standard method for representing management information is a desirable feature. The Common Information Model (CIM) [4] is an important standardization effort for defining a model capable of representing management information. XACML does not directly support the creation of policies using CIM elements or any other standard model for reusing management information [8]. By the other hand, PCIMe-based models offers a straight-forward way for creating policies that refers to CIM objects by using "*PolicyExplicitVariables*" (see section 5).

The "Ponder Language" is another important contribution in the policy-based management domain [13]. Different from XACML, which addresses only the access control problem, Ponder can be applied to a wide range of management domains. The Ponder language supports distinct types of policies. Authorization policies define the actions that subjects are permitted (or forbidden) to perform on target objects when certain conditions are satisfied. Obligation policies define the actions that subjects must perform on target objects when certain events occur. Composite policies provide facilities for grouping policies and structure them according to the organizational structure or other management needs. The Ponder project is continuously evolving, and recent Ponder publications starts exploring the use of CIM as the model for representing the policy target mechanisms and capabilities for *diffserv* frameworks [13]. The strategy adopted for Ponder implementation is quite different from the work described in this paper. The Ponder framework can be implemented by using a toolkit which permits to generate Java classes for building policy decision and policy enforcement objects. The framework also supports a strategy for notifying events to the policy objects responsible for supporting obligation policies. By the other hand, our proposal, describes access control policies as a PIB, which is provisioned to the PEP using the COPS-PR protocol. In the PIB, the RBAC policy elements are distinctly identified, offering a flexible method for updating the PIB and notifying information to the PDP. Also, because no assumption is made about the PEP implementation, the RBAC PIB information can be explored by applications or PEPs under distinct strategies.

The work "Role-Based Access Control for XML Enabled Management Gateways" [14] defines a XML/SNMPv3 gateway for RBAC. The RBAC policy is defined in XML (using an schema proposed by the authors). The Gateway is responsible for mapping the RBAC XML-policy to a MIB structure and configuring the network devices using SNMPv3. The authors also discusses the advantages of using RBAC policies to simplify the management of network devices. The RBAC-PIB, proposed in our paper, is also represented in XML, but follows the rigid structure defined by the framework PIB [9]. However, a similar approach as described in Cridlig et al [14] could be used for applying the RBAC policy network devices, i.e., creating a RBAC-PIB/SNMPv3 gateway. Conceptually, this gateway could be considered as part of a PEP.

There are also tool kits availables for simplifying the process for building COPS-PR based frameworks [15]. The white paper [15] also brings an interesting discussion about the advantages of applying the provisioning approach for several management domains, including access control.

4. The RBAC Provisioning Framework

Figure1 illustrates the three main elements in the RBAC provisioning framework: the policy server, the policy client and the policy repository. The policy server (i.e., the PDP) is the entity responsible for interpreting and distributing the policy information to the policy clients. A PEP can be considered the component in the policy client responsible for communicating with the PDP and supplying local policy decisions or installing the configuration into the device. The communication between the PEP and the PDP is implemented by the standard COPS-PR protocol. Figure 1 illustrates only one PEP, but a single PDP in the provisioning approach can handle a large number of PEPs. The performance issue is addressed in the section 7. As suggested by IETF, the policy and CIM information are both mapped to a LDAP schema. Because LDAP supports remote references through its schema, the CIM and LDAP repository are not required to be implemented in the same LDAP server. Also, both, policy and CIM information could be implemented using other technology, such as XML.

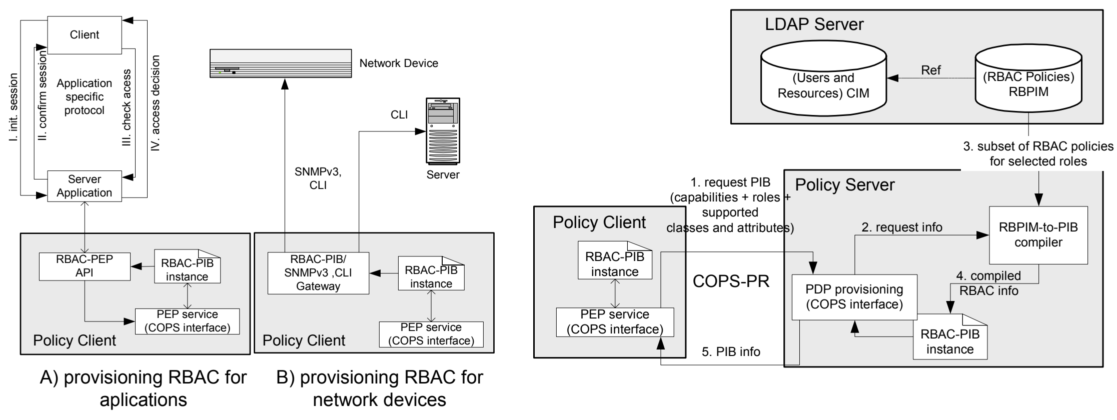

Client
I. init. session
II. confirm session
Application specific protocol
III. check acess
IV. access decision
Server Application
RBAC-PEP API
RBAC-PIB instance
PEP service (COPS interface)
Policy Client
Network Device
CLI
Server
SNMPv3, CLI
RBAC-PIB/ SNMPv3 ,CLI Gateway
RBAC-PIB instance
PEP service (COPS interface)
Policy Client
A) provisioning RBAC for aplications
B) provisioning RBAC for network devices
LDAP Server
(Users and Resources) CIM
Ref
(RBAC Policies) RBPIM
3. subset of RBAC policies for selected roles
1. request PIB (capabilities + roles + supported classes and attributes)
Policy Server
Policy Client
RBAC-PIB instance
PEP service (COPS interface)
COPS-PR
2. request info
RBPIM-to-PIB compiler
4. compiled RBAC info
PDP provisioning (COPS interface)
RBAC-PIB instance
5. PIB info

Figure 1. RBAC Provisioning Framework Overview

The RBAC-PIB information can be explored by two approaches, as illustrated in Figure 1. In the first approach (A), a PEP represents a server application which can be responsible for serving a large number of clients. The communication protocol between the server application and its clients is not imposed by our framework. In our current implementation, the server application communicates with the RBAC framework through a set of RBAC-based API, which follows the definitions proposed by the NIST standard [1]. These API are described in details in Nabhen et al [12]. In the second approach (B) the information in the RBAC-PIB is translated to configuration commands to the underlying system or to network devices via SNMPv3 or CLI – Command Line Interface.

The typical sequence of events related to policy provisioning and a PEP decision is also illustrated in Figure 1 (the explanation in this section follows the numbers in the arrows in the figure). When initialized, the PEP establishes a COPS-PR connection to the PDP, and requests an initial policy provisioning (i.e., a “full state” request) (1). As defined by IETF, the PEP supply in the policy request message a combination of “roles+capabilities” that are used to select a sub-set of policies that are required by the application(s) or device(s) interface(s) the PEP represents (e.g., “Warehouse Server” or “DMZ firewall Inbound Interface”). On receiving the request, the PDP activates the

RBPIM-to-PIB compiler in order to generate a RBAC-PIB for the PEP (2). The RBPIM-to-PIB compiler collects the subset or RBAC policies associated to the selected "roles" (3) and compiles the information into a RBAC-PIB (4). The RBPIM-to-PIB transformation is described in section 6. The PDP returns the PIB information to the PEP, using the COPS-PR protocol (5). The PEP stores the PIB information in a local repository.

Note, in the figure, that the PDP also keeps a copy of the RBAC-PIB in memory. This is required because the COPS-PR is a stateful protocol and PEP information is required in order to restore the PEP information in case of failure or to update information in the PEP.

5. RBPIM

This section discusses the RBPIM model. As defined in Ferraiolo et al [1], the RBAC model includes sets of five basic data elements called users (USER), roles (ROLES), objects (OBS), operations (OPS), and permissions (PRMS). The main idea behind the RBAC model is that permissions are assigned to roles instead of being assigned to users. The User Assignment (UA) is a many-to-many relationship (i.e., a user can be assigned to one or more roles, and a role can be assigned to one or more users). The Permission Assignment (PA) is also a many-to-many relationship (i.e., a permission can be assigned to one or more roles, and a role can be assigned to one or more permissions). A permission is an approval to perform an operation (e.g., read, write, execute, etc.) on one or more RBAC protected objects (e.g., a file, directory entry, software application, etc.). Role hierarchies define an inheritance relation of permissions among roles. The Static Separation of Duty (SSD) model element introduces static constraints to the User Assignment (UA) relationship by excluding the possibility of the user to assume conflicting roles. An important concept in RBAC is that roles must be activated in a session. The Dynamic Separation of Duty (DSD) model element introduces constraints on the roles a user can activate within a session.

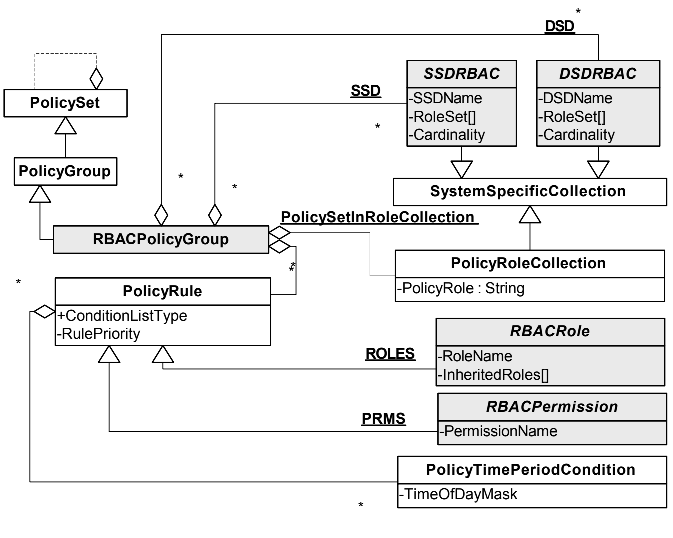


Figure 2. RBPIM Classes and Associations

The RBPIM (Role-Based Policy Information Model) is a PCIM extension for describing access control policies based on RBAC. RPBIM adopts the RBAC model [1], but some extensions have been introduced in order to provide a more flexible method for mapping users to roles and describing permissions and also for establishing network topology-based and time-based permission constraints. Figure 2 shows the revised RBPIM model adapted to the provisioning approach. The gray classes were introduced by the RBPIM model. The others are defined by PCIM/PCIMe [5,6] and CIM Core[4].

The *RBACPolicyGroup* defines a set of policy information that must be considered when generating a *RBAC-PIB*. Usually, in a large distributed environment, the policy repository will contain a large number of *RBACPolicyGroup* instances, each one associated with one or more *PolicyRoleCollection*. When a PEP requests the policy provisioning to the PDP, it supplies the "roles" assigned to its interface(s). By using the *PolicySetInRoleCollection* association, the PDP selects only the *RBACPolicyGroup* instances that must be considered for that particular interface.

Basically, RBPIM introduces two *PolicyRule* extensions, named *RBACRole* (representing roles ∈ ROLES) and *RBACPermission* (representing permissions ∈ PRMS). The *PolicyTimePeriodCondition* instances are used for imposing time constraints to the use of roles and permissions. The static and dynamic separation of duty constraints are represented, respectively, by DSDRBAC and SSDRBAC instances, according to the semantic described [1]. Both classes are specializations of "*SystemSpecificCollection*", defined by the CIM Core. Note that the SSD and DSD constraints are imposed to the *RBACPolicyGroup*. Therefore, they could not be represented as rule conditions.

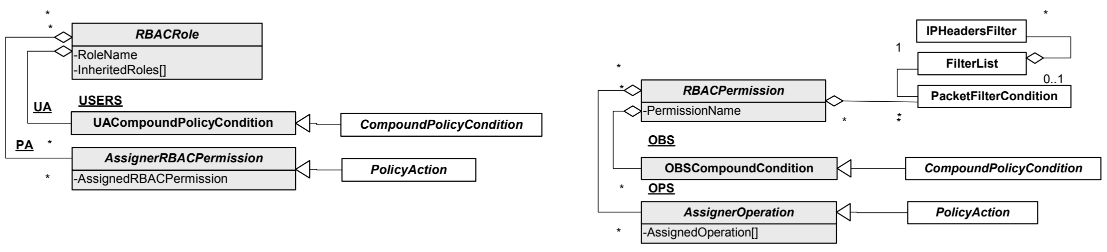


Figure 3. RBACRole and RBACPermission

The *RBACRole* class and its associations are illustrated in Figure 3. Using the *PolicyRule* semantics defined by PCIM, a *RBACRole* instance express the following rule: "*If conditions are satisfied than assign the RBACRole permission(s) to the user(s)*". As shown in the Figure 3, users ∈ USERS are represented by a *CompoundPolicyCondition* extension, called *UACompoundPolicyCondition*. The use of the *CompoundPolicyCondition* semantics simplifies the process of assigning a role to a user (UA) because the assignment can be implemented with predefined CIM information about the users and organization. For more details, please, see references [6] and [12]. The permission is defined by a *PolicyAction* extension called *AssignerRBACPermission.*

The *RBACPermission* class and its associations are illustrated in Figure 3. A *RBACPermission* instance express the following rule: "*If conditions are satisfied than assign the operation permission(s) to the object(s)*". Objects ∈ OBS are defined by the *OBSCompoundCondition* instances. Again, by defining an expression that combines attributes of objects already described in the CIM repository, the use of a

CompoundCondition simplifies the process of defining permissions. The *AssignerOperation* instances are used to represent operations ∈ OPS. The *PacketFilterConditions* are used to restrict the permissions according to the network topology.

Finally, note that because *RBACRole* and *RBACPermission* are both *PolicyRule* extensions, they are indirectly associated by matching the attribute *AssignedRBACPermission* (from *AssignerRBACPermission*) with the attribute *PermissionName* (from *RBACPermission*).

The example in Figure 4 illustrates the use of the RBPIM model. The *RBACRole* in the figure was called "Auditor". The attribute *InheritedRoles* is used for expressing the Hierarchical RBAC, i.e., the role "Auditor" inherits the permissions of role "Employee". The UA relationship for "Auditor" points to a compound condition with a single simple condition, based on a *PolicyExplicityVariable*. The explicit variable permits to create conditions referring to CIM objects. In this case, the "Auditor" role is assigned to all users where the *BusinessCategory* attribute match "C1". The UA assignment is restricted to the period between 10h00 and 16h00 by the *PolicyTimePeriodCondition* instance.

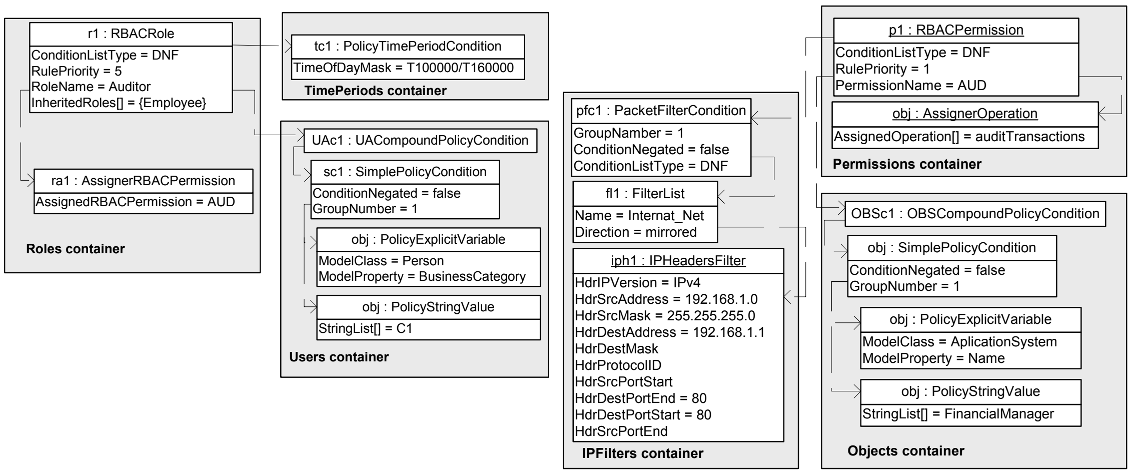


Figure 4. RBPIM policy example

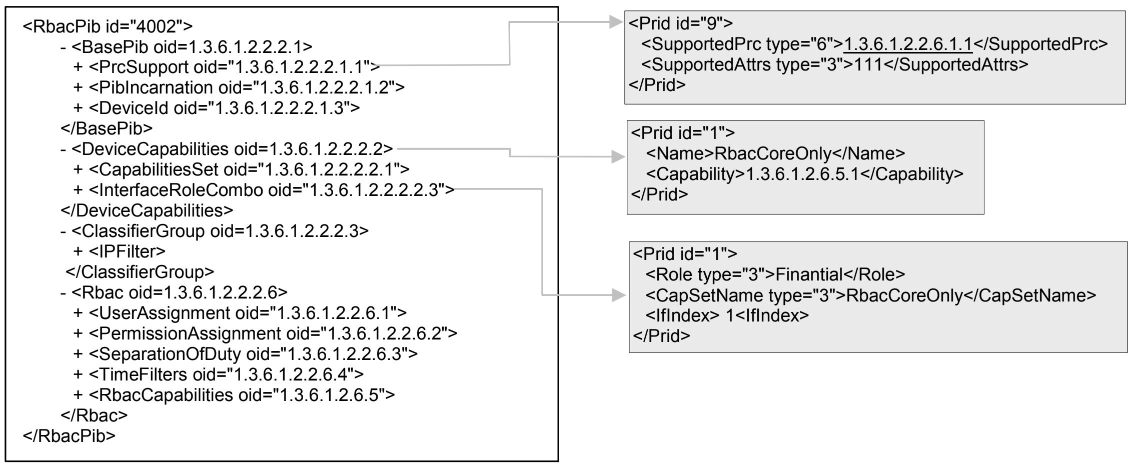
The "Auditor" has a PA relationship with a permission called "AUD". This permission defines that the operation "*auditTransactions*" can be executed when *OBSCompoundConditions* and the *PacketFilterCondition* are simultaneously satisfied. In this case, the compound condition includes an explicit variable condition pointing to a CIM object that represents an specific application. The *PacketFilterCondition* restricts the operation from machines within the 192.168.1.0/24 subnet.

As well as PCIM, the RBPIM model has a neutral implementation. RBPIM mapping to LDAP schema has been implemented according to the IETF standard PCLS [7]. In Figure 4, it is defined six containers where the policy objects are stored. In this approach, a container is created for storing "reusable" information. For example, "*RBACRole*" instances are stored in a "Roles" container. by the other hand, *AssignerRBACPermission* instances, are too simple for receiving a container, because duplicate objects is cheaper than pointing them. Hence, they are also stored in the

"Roles" container and associated to *RBACRole* instances by DIT containment. *UACompoundPolicyCondition* instances is also a worthy reusable information and, therefore, are stored in a specific container. *RBACRoles* instances have received the required attributes for pointing to the *UACompoundCondition* instances and grouping then according to a DNF or CNF strategy. The same reasoning applies to the other classes in the figure, i.e., worthy reusable information receives a container and other associations are implemented by DIT containment.

6. RBAC-PIB

This work defines a RBAC-PIB which represents the information transferred from the PDP to the PEP during provisioning process. The RBAC-PIB is based on the IETF PIB-framework definitions [9]. Figure 4 shows the RBAC-PIB structure represented in XML. A PIB can be described as a conceptual tree namespace where the branches of the tree represent structures of data or Provisioning Classes (PRCs), while the leaves represent various instantiations of Provisioning Instances (PRIs). The PRCs corresponding to the *BasePib*, *DeviceCapabilities* and *ClassifierGroup* groups are defined by the Framework PIB [9]. All PIB elements corresponding to branches have an "oid" attribute, defined according to RFC 3159. The oid prefix 1.3.6.1.2.2.2 refers to the framework PIB definition. The PRCs corresponding to the Rbac group are extensions defined by our proposal. The oid prefix 1.3.6.1.2.2.2.6, currently unused, have been assigned to identify the PRC classes corresponding to the Rbac information.



```
<RbacPib id="4002">
      - <BasePib oid=1.3.6.1.2.2.2.1>
         + <PrcSupport oid="1.3.6.1.2.2.2.1.1">
         + <PibIncarnation oid="1.3.6.1.2.2.2.1.2">
         + <DeviceId oid="1.3.6.1.2.2.2.1.3">
      </BasePib>
      - <DeviceCapabilities oid=1.3.6.1.2.2.2.2>
         + <CapabilitiesSet oid="1.3.6.1.2.2.2.2.1">
         + <InterfaceRoleCombo oid="1.3.6.1.2.2.2.2.3">
      </DeviceCapabilities>
      - <ClassifierGroup oid=1.3.6.1.2.2.2.3>
         + <IPFilter>
       </ClassifierGroup>
      - <Rbac oid=1.3.6.1.2.2.2.6>
         + <UserAssignment oid="1.3.6.1.2.2.6.1">
         + <PermissionAssignment oid="1.3.6.1.2.2.6.2">
         + <SeparationOfDuty oid="1.3.6.1.2.2.6.3">
         + <TimeFilters oid="1.3.6.1.2.2.6.4">
         + <RbacCapabilities oid="1.3.6.1.2.6.5">
      </Rbac>
</RbacPib>
```

```
<Prid id="9">
  <SupportedPrc type="6">1.3.6.1.2.2.6.1.1</SupportedPrc>
  <SupportedAttrs type="3">111</SupportedAttrs>
</Prid>
```

```
<Prid id="1">
  <Name>RbacCoreOnly</Name>
  <Capability>1.3.6.1.2.6.5.1</Capability>
</Prid>
```

```
<Prid id="1">
  <Role type="3">Finantial</Role>
  <CapSetName type="3">RbacCoreOnly</CapSetName>
  <IfIndex> 1<IfIndex>
</Prid>
```

Figure 5. RBAC-PIB structure represented in XML.

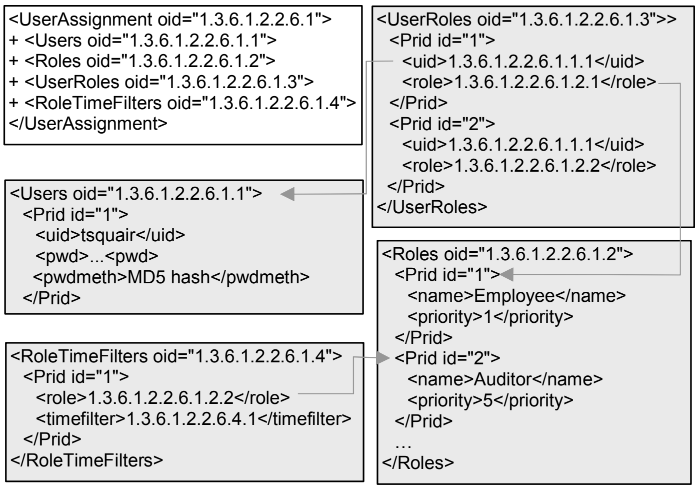
A PRC can be used for both supplying ("*notify*") or receiving ("*install*") information to/from the PDP. A PRC can be also "*install-notify*", providing bidirectional exchange of information between the PEP and the PDP. The *BasePib* have three tables grouping the instances of the PRCs named *PrcSupport*, *PibIncarnation* and *DeviceId*. All classes are "notify" except *PibIncartion,* which is "install-notify". The *PrcSupport* instances define the classes and attributes supported by the PIB. As example, Figure 5 shows the PRI (instance with id="9") corresponding to the PRC *User* (in the *UserAssignment* element of the *Rbac* group). The *SupportedAttrs* attribute is a binary map indicating the PEP supports all three attributes defined for the class. This information is used by the PDP, in order to determine which attributes must be transferred to the PEP during the provisioning process.

The *PibIncarnation* instance (this PRC contains exactly one row) includes information about the PDP, the version of the policy currently downloaded and the behavior of the PEP when the connection with the PDP is closed. An attribute called <*FullState*> plays an important role in the provisioning process. The PEP use *FullState=true* for asking a full state update to the PDP (in this case, any previous state in the PDP is erased) and *FullState=false* for asking an incremental update. *DeviceId* supplies additional information for the PDP to identify the PEP (e.g., RBAC version or model).

DeviceCapabilities group supplies information to the PDP permitting to select and, eventually, adapt the policies to be provisioned to the PEP. The *CapabilitiesSet* defines optional information about specific mechanisms supported by the PEP. The *CapabilitiesSet* defines pointers to specific capabilities defined by the *RbacCapabilities* section in the *RbacGroup* (explained further in this section). The *InterfaceRoleCombo* instances indicate the roles and capability sets that have been assigned to each interface of the managed element.

The *Classifier* contains the PRC <*IPFilter*>, permitting to describe filtering conditions based on the fields of the IP header. This PRC is used to represent the *IPHeadersFilter* conditions used in the RBPIM model for constraining the PA assignments.

The strategy adopted for defining the representation of the RBAC information follows the framework PIB definitions. PRC classes are used to group the RBAC information, all attributes are defined within PRI instances and pointers based on "oid's" are used to implement the association between classes. This approach is required in order to use the COPS-PR protocol for provisioning the policy information to the PEP. According to our proposal the RBAC group contains five elements: <*UserAssignment*>, <*PermissionAssignment*>, <*SeparationOfDuty*>, <*TimeFilters*> and <*RbacCapabilities*>.



```
<UserAssignment oid="1.3.6.1.2.2.6.1">
+ <Users oid="1.3.6.1.2.2.6.1.1">
+ <Roles oid="1.3.6.1.2.2.6.1.2">
+ <UserRoles oid="1.3.6.1.2.2.6.1.3">
+ <RoleTimeFilters oid="1.3.6.1.2.2.6.1.4">
</UserAssignment>
```

```
<UserRoles oid="1.3.6.1.2.2.6.1.3">>
  <Prid id="1">
    <uid>1.3.6.1.2.2.6.1.1.1</uid>
    <role>1.3.6.1.2.2.6.1.2.1</role>
  </Prid>
  <Prid id="2">
    <uid>1.3.6.1.2.2.6.1.1.1</uid>
    <role>1.3.6.1.2.2.6.1.2.2</role>
  </Prid>
</UserRoles>
```

```
<Users oid="1.3.6.1.2.2.6.1.1">
  <Prid id="1">
    <uid>tsquair</uid>
    <pwd>...<pwd>
    <pwdmeth>MD5 hash</pwdmeth>
  </Prid>
```

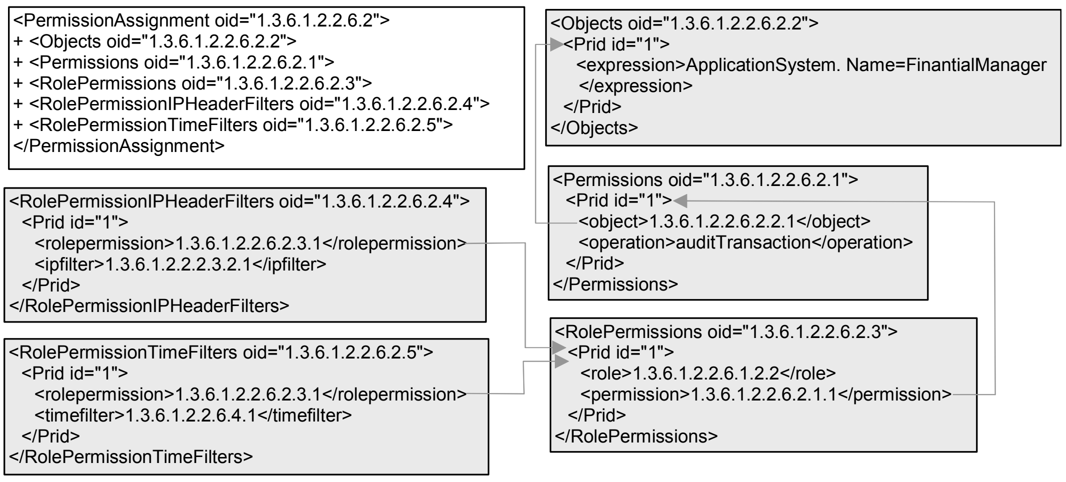
```
<Roles oid="1.3.6.1.2.2.6.1.2">
  <Prid id="1">
    <name>Employee</name>
    <priority>1</priority>
  </Prid>
  <Prid id="2">
    <name>Auditor</name>
    <priority>5</priority>
  </Prid>
  …
</Roles>
```

```
<RoleTimeFilters oid="1.3.6.1.2.2.6.1.4">
  <Prid id="1">
    <role>1.3.6.1.2.2.6.1.2.2</role>
    <timefilter>1.3.6.1.2.2.6.4.1</timefilter>
  </Prid>
</RoleTimeFilters>
```

Figure 6. Rbac PIB: UserAssignement Group

Figure 6 shows the structure <*UserAssignment*>, which is an element of the <*Rbac*> group. The PRIs in the figure correspond to the policy example described in Figure 4. The <*UserAssignment*> element contains four PRCs: <*Users*>, <*Roles*>, <*UserRoles*> and <*RoleTimeFilters*>. Each PRI in the <*Users*> PRC corresponds to a user identified by the <*uid*> attribute. The <*pwd*> and <*pwdmeth*> attributes are optionals (i.e., as informed by the <*PrcSupport*> structure). The authentication attributes are optionals because authentication management can be outside of the framework scope.

Similarly, each PRI in <*Roles*> corresponds to a RBAC role as defined by the *RBACRole* class in the RBPIM model. The UA assignment is defined by the <*UserRoles*> PRC, which is an association class between <*User*> and <*Roles*>. Note that all *UACompoundPolicyCondition* information in the RBPIM model is pre-processed by the PDP and the result is expressed by the <*UserRoles*> instances. During the process of defining the <*UserRoles*> PRIs, the PDP automatically creates the PRIs for representing the roles indirectly assigned to a user by heritage (by the *RBACRole.inheritedRoles[]* attribute in the RBPIM model). During the process of defining <*UserRoles*>, the PDP also takes into account the SSD constraints (corresponding to the *SSDRBAC* class in the RBPIM model) resulting that only the highest priority roles free of SSD constraints are assigned to a user in the PIB. Finally, the <*RoleTimeFilters*> is used to constraint the period a user can activate a role. Note that the time filter information refers to the <*TimeFilters*> PRC in the <*Rbac*> group structure in Figure 5.



```
<PermissionAssignment oid="1.3.6.1.2.2.6.2">
+ <Objects oid="1.3.6.1.2.2.6.2.2">
+ <Permissions oid="1.3.6.1.2.2.6.2.1">
+ <RolePermissions oid="1.3.6.1.2.2.6.2.3">
+ <RolePermissionIPHeaderFilters oid="1.3.6.1.2.2.6.2.4">
+ <RolePermissionTimeFilters oid="1.3.6.1.2.2.6.2.5">
</PermissionAssignment>
```

```
<Objects oid="1.3.6.1.2.2.6.2.2">
  <Prid id="1">
    <expression>ApplicationSystem. Name=FinantialManager
    </expression>
  </Prid>
</Objects>
```

```
<Permissions oid="1.3.6.1.2.2.6.2.1">
  <Prid id="1">
    <object>1.3.6.1.2.2.6.2.2.1</object>
    <operation>auditTransaction</operation>
  </Prid>
</Permissions>
```

```
<RolePermissionIPHeaderFilters oid="1.3.6.1.2.2.6.2.4">
  <Prid id="1">
    <rolepermission>1.3.6.1.2.2.6.2.3.1</rolepermission>
    <ipfilter>1.3.6.1.2.2.2.3.2.1</ipfilter>
  </Prid>
</RolePermissionIPHeaderFilters>
```

```
<RolePermissions oid="1.3.6.1.2.2.6.2.3">
  <Prid id="1">
    <role>1.3.6.1.2.2.6.1.2.2</role>
    <permission>1.3.6.1.2.2.6.2.1.1</permission>
  </Prid>
</RolePermissions>
```

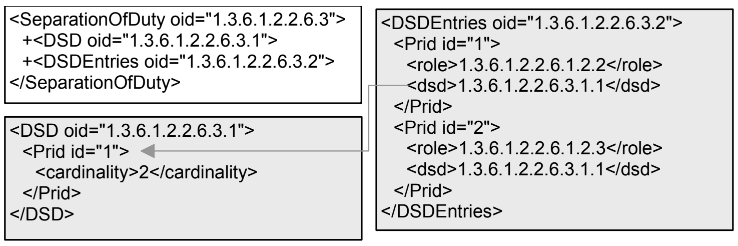
```
<RolePermissionTimeFilters oid="1.3.6.1.2.2.6.2.5">
  <Prid id="1">
    <rolepermission>1.3.6.1.2.2.6.2.3.1</rolepermission>
    <timefilter>1.3.6.1.2.2.6.4.1</timefilter>
  </Prid>
</RolePermissionTimeFilters>
```

Figure 7. RBAC PIB: Permission Assignment

Figure 7 shows the structure of the <*PermissionAssignment*> which is an element of the <*Rbac*> group. The <*PermissionAssignment*> element contains five PRCs: <*Objects*>, <*Permissions*>, <*RolePermissions*>, <*RolePermissionsIPHeadersFilters*> and <*RolePermissionTimeFilters*>. The <*Objects*> PRC defines the resources controlled by the RBAC policy. The resources are represented by CIM objects. Each <Objects> PRI contains a Boolean expression, formed by grouping policy explicit variables in CNF or DNF form. The <*Permissions*> PRC defines permissions by mapping an operation ("defined as a string attribute") to an <*Objects*> PRI. Alternatively, CIM offers also elements for describing operations that can be used in this approach by including the operation in the <*Objects*> expression. <*RolePermissions*> is the association class responsible for assigning permissions to RBAC roles. The permission assignment (PA) is constrained by the PRCs <*RolePermissionIPHeadersFilters*> and <*RolePermissionsTimeFilters*> permitting to define, respectively, subnet constraints and time period constraints to the permissions assigned to a role. Note that

<*RolePermissionIPHeadersFilters*> employees "oid" references to the <*IPFilter*> element defined by the Framework PIB.

The <*SeparationOfDuty*> element (see Figure 8) contains the RBAC definitions permitting the PEP to implement the dynamic separation of duty, i.e., constraints the roles a user can simultaneously activate within a section. The <*DSD*> PRC defines the DSD cardinality and the <*DSDEntries*> PRC defines the roles constrained by the DSD. Note that the static separation of duty constraints are pre-processed by the PDP and, therefore, are not included in the PIB.



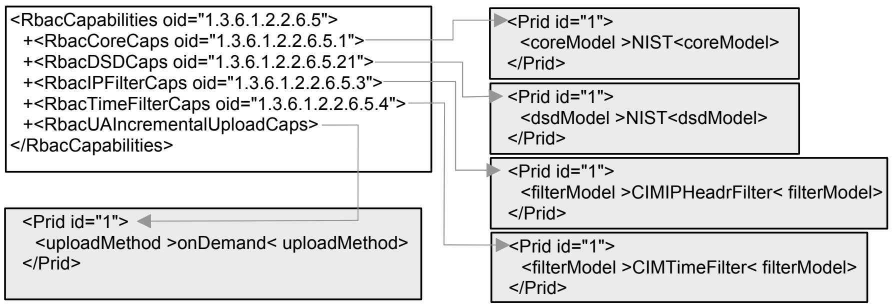
```
<SeparationOfDuty oid="1.3.6.1.2.2.6.3">
  +<DSD oid="1.3.6.1.2.2.6.3.1">
  +<DSDEntries oid="1.3.6.1.2.2.6.3.2">
</SeparationOfDuty>
```

```
<DSD oid="1.3.6.1.2.2.6.3.1">
  <Prid id="1">
    <cardinality>2</cardinality>
  </Prid>
</DSD>
```

```
<DSDEntries oid="1.3.6.1.2.2.6.3.2">
  <Prid id="1">
    <role>1.3.6.1.2.2.6.1.2.2</role>
    <dsd>1.3.6.1.2.2.6.3.1.1</dsd>
  </Prid>
  <Prid id="2">
    <role>1.3.6.1.2.2.6.1.2.3</role>
    <dsd>1.3.6.1.2.2.6.3.1.1</dsd>
  </Prid>
</DSDEntries>
```

Figure 8. RBAC PIB: Separation of Duty

<*RbacCapabilities*> contains the elements pointed by the <*CapabilitiesSet*> from the framework PIB. It is composed by 5 elements (see Figure 9). <*RbacCoreCaps*> defines the support to the basic access control functionalities, as defined by the NIST. Presently, only the NIST model is supported, but future extensions could include alternative models. <*RbacDSDCaps*> defines the support to dynamic separation of duty constraints. Usually, network devices do not have support to this functionality. <*RbacIPFilterCaps*> and <*RBacTimeFilterCaps*> define the support to network and time constraints imposed to *Rbac* permissions. These features are not presented in the NIST specification, being extensions proposed by the RBPIM. In our current implementation, when the constraining capabilities (DSD, *Time* and *IPFilter*) are not supported the corresponding *Permission* and *Roles* are simply eliminated.



```
<RbacCapabilities oid="1.3.6.1.2.2.6.5">
  +<RbacCoreCaps oid="1.3.6.1.2.2.6.5.1">
  +<RbacDSDCaps oid="1.3.6.1.2.2.6.5.21">
  +<RbacIPFilterCaps oid="1.3.6.1.2.2.6.5.3">
  +<RbacTimeFilterCaps oid="1.3.6.1.2.2.6.5.4">
  +<RbacUAIncrementalUploadCaps>
</RbacCapabilities>
```

```
<Prid id="1">
  <uploadMethod >onDemand< uploadMethod>
</Prid>
```

```
<Prid id="1">
  <coreModel >NIST<coreModel>
</Prid>
```

```
<Prid id="1">
  <dsdModel >NIST<dsdModel>
</Prid>
```

```
<Prid id="1">
  <filterModel >CIMIPHeadrFilter< filterModel>
</Prid>
```

```
<Prid id="1">
  <filterModel >CIMTimeFilter< filterModel>
</Prid>
```

Figure 9. RBAC Capabilities

Finally, <*RbacIncrementalUploadCaps*> defines an optional framework feature. When this feature is present, the PIB information generated in the initial provisioning process is not complete because it lacks the UA assignment (i.e., the mapping between user and roles). The UA assignment is not initially provisioned because an application with a large number of potential users would lead to a extremely large PIB. Instead, the UA assignment is incrementally uploaded to the PIB when a new RBAC session is created. The PEP requests the UA assignment to the PDP using the COPS-PR protocol (*FullState*=false) and receives only the PIB elements concerning the UA assignment of the new user. After this event, the check access requests can be locally decided by the PEP. Note that this feature is not useful when the PIB is used for generating configuration commands (the traditional PIB approach), it applies only to RBAC-aware applications.

7. Evaluation

Our proposal has been evaluated in terms of two criteria: a) the size of the PIB with respect to the complexity of the RBAC policy, i.e., the number of policy elements (Roles, PRMS, OBJS, OPS, DSD and SSD, as defined in session 5) that must be processed in order to generate the PIB; b) the time required for provisioning a PIB. The provisioning time includes the time for compiling the RBPIM model, generating the PIB and transferring it to the PEP using the COPS-PR protocol.

The PDP has been implemented in Java and runs in a Pentium IV 1.6 GHz, 1GB memory PC. The policies are stored in a *OpenLdap* server, version 2.7. The PDP/PEP are connected by a 100 Mbps Ethernet LAN. The evaluated scenario corresponds to provisioning RBAC for applications (see item *A* in Figure 1). In this scenario, we assume a managed device with support to the <*RbacIncrementalUploadCaps*), i.e., user information is not initially provisioned. Table 1 summarizes the results of the provisioning time evaluation for a subset of RBAC policies that concerns "a single PIB" (as defined by the interface roles of the managed element). Note in the Table 1, the effect of the number of RBAC objects in the PIB size and provisioning time. The time for provisioning "one" user is also presented in the table. The PIB size corresponds to the XML representation adopted in this paper.

Table 1. Evaluation of Provisioning Time

Roles	PRMS	OBJS	OPS	DSD	SSD	Initial Provisioning (ms)	PIB size	Provisioning a user (ms)	PIB Size (+user)
10	6	3	8	1	3	2133	16k	0280	17k
40	6	3	8	1	3	3775	34k	0519	39k
80	6	3	8	1	3	6460	58k	1111	68k
20	10	7	12	1	3	2974	24k	0370	24k
20	40	37	42	1	3	4256	42k	0441	42k
20	80	77	82	1	3	7180	67k	0410	67k
20	6	3	8	10	3	2845	25k	0451	25k
20	6	3	8	40	3	2895	36k	0391	36k
20	6	3	8	80	3	3655	51k	0431	51k
20	6	3	8	1	10	2864	22k	0421	22k
20	6	3	8	1	40	2724	22k	0441	22k
20	6	3	8	1	80	2684	22k	0411	22k
Obs. The initial Provisioning corresponds to a time used to load every object in the PIB except the UA Assignment									

8. Conclusion

This paper has presented a policy based framework for implementing RBAC policies in heterogeneous and distributed systems adopting a provisioning approach. The framework has been implemented in accordance with the IETF standards and a superset of the NIST RBAC standard. This work has proposed a RBAC-based information model and a RBAC-based PIB. These elements combined with the COPS-PR protocol offers a flexible method for distributing and updating policies in distributed and heterogeneous systems. Future works include extending the provisioning approach for other access control languages and building an SNMPv3 gateway for the RBAC PIB.

9. References

[1] D.F. Ferraiolo, R.S. Sandhu, G. Serban; "A Proposed Standard for Role-Based Access Control", *ACM Transactions on Information System Security*, Vol. 4, No. 3, (2001) pp. 224-274".

[2] J. Schnizlein, J. Strassner, M. Scherling, B. Quinn, S. Herzog, A. Huynh, M. Carlson, J. Perry, S. Waldbusser; "Terminology for Policy-Based Management", IETF RFC 3198, Nov. 2001.

[3] K. Chan, J. Seligson, D. Durham, S. Gai, K. McCloghrie, S. Herzog, F. Reichmeyer, R. Yavatkar, A. Smith, "COPS Usage for Policy Provisioning (COPS-PR)", IETF RFC 3084, Mar. 2001.

[4] Distributed Management Task Force (DMTF); "Common Information Model (CIM) Specification", URL: http://www.dmtf.org (2003).

[5] B. Moore, E. Elleson, J. Strasser, A. Weterinen; "Policy Core Information Model", IETF RFC 3060, Feb. 2001.

[6] B. Moore, E. Elleson, J. Strasser, A. Weterinen; "Policy Core Information Model Extensions"; IETF RFC 3460, Feb. 2001.

[7] J. Strassner, E. Ellesson, B. Moore, R. Moats. "Policy Core Lightweight Directory Access Protocol (LDAP) Schema", IETF RFC 3707, Feb. 2004.

[8] E. Toktar, E. Jamhour, C. Maziero, "RSVP Policy Control using XACML", *IEEE 5th International Workshop on Policies for Distributed Systems and Networks. (POLICY 2004)*, New York, 2004, pp. 87-98.

[9] R. Sahita, S. Hahn, K. Chan, K. McCloghrie; "Framework Policy Information Base", IETF RFC 3318, Mar. 2003.

[10] OASIS: "eXtensible Access Control Markup Language (XACML)", version 1.03. OASIS Standard, Feb. 2003, URL: http://www.oasis-open.org

[11]. OASIS; "XACML Profile for Role Based Access Control (RBAC)", draft, Feb. 2004, URL: http://www.oasis-open.org

[12] R. Nabhen, E. Jamhour, C. Maziero, "Policy-Based Framework for RBAC", *14th IFIP/IEEE International Workshop on Distributed Systems: Operations & Management, Germany (DSOM 2003),* Oct. 2003, pp. 181-193.

[13] L. Lymberopoulos, E. C. Lupu, M. S. Sloman, "Ponder Policy Implementation and Validation in a CIM and Differentiated Services Framework", *NOMS 2004*, Seoul, Apr. 2004.

[14] V. Cridlig, O.Festor, R.State, "Role-Based Access control for XML Enabled Management Gateways", *15th IFIP/IEEE International Workshop on Distributed Systems: Operations & Management* (DSOM 2004), Davis, USA, Nov. 2004, pp. 183-195.

[15] R. Fenger, H. Hegde, D. Larson, R. Sahita, "Simplifying Support of New Network Services Using COPS-PR", *Intel Corporation white paper*, 2002, URL http://www.intel.com/labs/manage.cops.