RBPIM: Enforcing RBAC policies in distributed heterogeneous systems

Ricardo Nabhen, Edgard Jamhour, Carlos Maziero

PPGIA, Pontifícia Universidade Católica do Paraná (PUC-PR)
Rua Imaculada Conceição, 1155 – Prado Velho – Curitiba – PR

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

Abstract. *This paper presents a PCIM-based framework for storing and enforcing RBAC (Role Based Access Control) policies in distributed heterogeneous systems. PCIM (Policy Core Information Model) is an information model proposed by IETF. PCIM permits to represent network policies in a standard form, allowing software from different vendors to read the same set of policy rules. This paper describes a PCIM extension, called RBPIM (Role-Based Policy Information Model), in order to represent network access policies based on the RBAC model. A RBPIM implementation framework based on the PDP/PEP (Policy Decision Point/Policy Enforcement Point) approach is also presented and evaluated.*

1. Introduction

Policy-based networking (PBN) is a management approach developped for simplifying network administration. In PBN, a policy is a formal set of statements that define how network resources are allocated among its clients. In order to implement PBN it is important to define a vendor independent method for representing and storing policies and network resources. An important work in this field, called CIM (Common Information Model), was proposed by the DMTF [Distributed Management Task Force 1999]. The CIM model addresses the problem of representing network resources. PCIM (Policy Core Information Model) is an information model proposed by IETF that extends CIM classes in order to support policy definitions for managing these resources [Moore, B. 2001]. PCIM is a generic policy model. Application-specific areas must be addressed by extending the policy classes and associations proposed by PCIM. For example, QPIM (QoS Policy Information Model) is a PCIM extension for describing quality of service polices [Snir, Y., 2001]. In this context, this paper describes a PCIM extension for access control, called RBPIM (Role Based Policy Information Model), which permits to represent network access control policies based on roles, as well as static and dynamic constraints, as defined by the proposed NIST RBAC standard [Ferraiolo, D.F., 2001]. Typically, PCIM is implemented using a PDP/PEP approach [Yavatkar, R., 2000]. The PDP (Policy Decision Point) is a network policy server responsible for supplying policy information for network devices and applications. The PEP (Policy Enforcement Point) is the policy client (usually, a component of the network device/application) responsible for enforcing the policy. The communication between the PDP and the PEP is implemented by the COPS protocol, defined by the IETF [Durham, Ed., 2000].

This paper extends our previous papers [Nabhen, R., 2003,1], [Nabhen, R., 2003,2] and [Nabhen, R., 2003,3], with the inclusion of the RBPIM LDAP Mapping (section 5) and a new version of the RBPDP algorithms (section 6.3), a more formal version following the NIST proposed standard notation. The remaining of this paper is organized as follows: Section 2 describes the RBAC model used in this paper. Section 3 reviews some related works. Section 4 presents the RBPIM information model. Section 5 presents the RBPIM LDAP Schema for using LDAP-based directory services as RBPIM policy repositories. Section 6 presents the RBPIM framework implemented using the outsourcing model, as defined by the COPS standard. Section 7 presents the performance evaluation results of a prototype of the RBPIM framework under various load conditions. Finally, the conclusion summarizes the main aspects in this project and points to future works.

2. RBAC Model

RBAC models have received a broad support as a generalized approach to access control, and are well recognized for their many advantages in performing large-scale authorization control. The RBAC model adopted by the RBPIM framework is based on proposed NIST (National Institute of Standards and Technology) Standard [Ferraiolo, D.F., 2001]. The PEP implementation in the RBPIM framework (called RBPEP – Role Based PEP) is based on APIs described in the proposed NIST RBAC functional. The proposed NIST standard presents a RBAC reference model based on four components: Core RBAC, Hierarchical RBAC, Static Separation of Duty Relations and Dynamic Separation of Duty Relations. For a more complete description, please, refer to the proposed NIST standard [Ferraiolo, D.F., 2001]. The RBPIM framework described in sections 4, 5 and 6 supports all four elements of the proposed NIST standard and proposes a more flexible method for defining UA relationships by combining the explicit and implicit variables of the PCIMe model [Moore, B., 2003].

3. Related Work

Recent works starts exploring the advantages of the PDP/PEP approach for implementing an authorization service that could be shared across a heterogeneous system in a company. For example, the XACML (eXtensible Access Control Markup Language), proposed by the OASIS consortium [OASIS 2003], is a XML based language that describes both an access control policy language and a request/response language. The request/response language is used for supporting the communication between PEP clients and PDP servers. The RBPIM framework described in this paper also uses the PDP/PEP approach. However, our approach differs from XACML because (1) the RBPIM uses a standard COPS protocol for supporting the PEP/PDP communication, (2) the information model used for describing policies is based on a PCIM extension and (3) RBPIM has been implemented for supporting RBAC.

Most of the research efforts found in the literature refer to the use of the PCIM model and its extensions for developing policy management tools for QoS support [Snir, Y., 2001]. However, a pioneer work for defining a PCIM extension for supporting RBAC, called CADS-2, has been proposed by BARTZ, L.S. [Bartz, L. 2001]. The CADS-2 is a review of a previous work, called *hyperDRIVE*, also proposed by BARTZ [Bartz, L. 1997]. The *hyperDRIVE* is a LDAP schema for representing RBAC. This schema can be considered as a first step for implement RBAC using the

PDP/PEP approach. As *hyperDRIVE*, CADS-2 defines classes suitable to be implemented in a directory-based repository, such as LDAP. The RBPIM model described in the section 4 uses some ideas presented in the CADS-2 model, as the idea of mapping roles to users using Boolean expressions. Note that this approach offers an additional degree of freedom for creating RBAC policies because the UA (*User Assingment*) relationship can be expressed through Boolean expressions instead of a direct mapping between user and roles. However, the IETF publication PCIMe (PCIM Extensions) proposes a different approach for representing Boolean expressions [Moore, B., 2003]. The RBPIM framework adopts the PCIMe strategy. Also, many features have been introduced in order to support the other elements of the RBAC model, such as hierarchy of roles, DSD and SSD, not supported in the original CADS-2 model.

4. RBPIM: The Role-Based Policy Information Model

The RBPIM model is a PCIM extension for representing RBAC policies. The RBPIM class hierarchy is shown in the Figure 1. The following classes have been introduced: *RBACPermission* and *RBACRole* (specializations of *PolicyRule*), *AssignerPermission* and *AssignerOperation* (specializations of *PolicyAction*), *DSDRBAC* and *SSDRBAC* (specializations of *Policy*). The *RBACPolicyGroup* class is used to group the information of the constrained RBAC model. The RBPIM model uses the *SimplePolicyCondition* specializations as proposed by PCIMe.

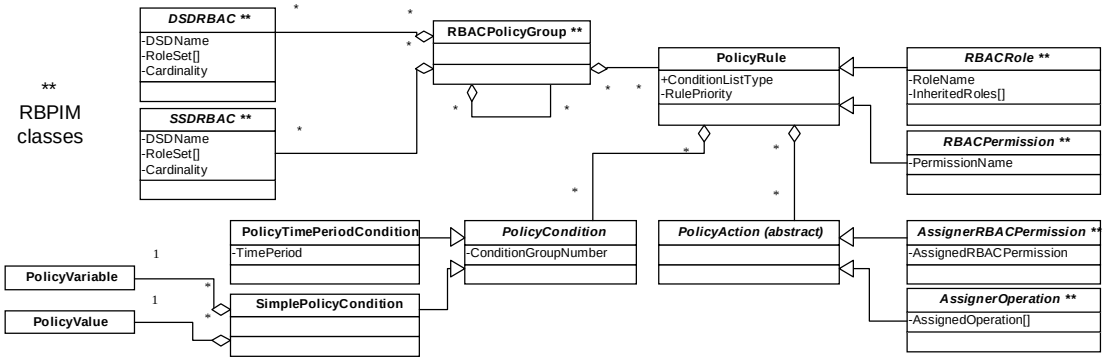


Figure 1. RBPIM class hierarchy.

As shown in Figure 2, the approach in the RBPIM model consists in using two specializations of *PolicyRule* for building the RBAC model: *RBACRole* (for representing RBAC roles) and *RBACPermission* (for representing RBAC permissions). *RBACRole* can be associated to lists of *SimplePolicyCondition*, *AssignerRBACPermission* and *PolicyTimePeriodCondition* instances. The instances of *SimplePolicyCondition* are used to express the conditions for a user to be assigned to a role (UA relationship). The instances of *AssignerRBACPermission* are used to express the permissions associated to a role (PA relationship). The instances of *PolicyTimePeriodCondition* define the periods of time a user can activate a role. *RBACPermission* can be associated to a list of *SimplePolicyCondition* and *AssignerOperation* instances. The instances of *SimplePolicyCondition* are used to describe the protected RBAC objects and the instances of *AssignerOperation* are used to describe approved operation on these

objects.

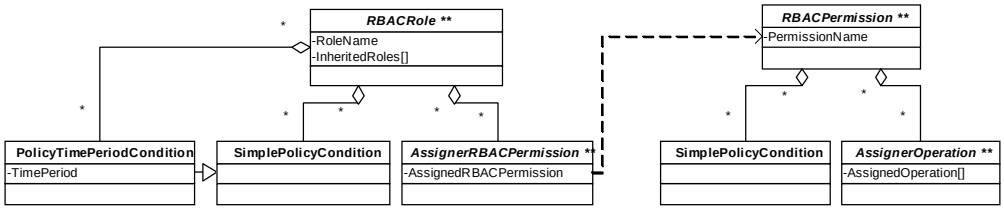


Figure 2. RBPIM class associations.

The example in Figure 3 to illustrates the use of the RBPIM model. The *RBACRole* in the figure was called "role1". The attribute *InheritedRoles* is used for expressing the Hierarchical RBAC, i.e., the role "role 1" inherits the permissions of roles "role2" and "role3". The UA relationship for "role1" is defined as: IF "*PolicySourceIPv4Variable* MATCH 192.168.10.0/24" AND "*Person.BusinessCategory* MATCH CT*" AND "*PolicyTimePeriodCondition* MATCH [20020701,20031201]".

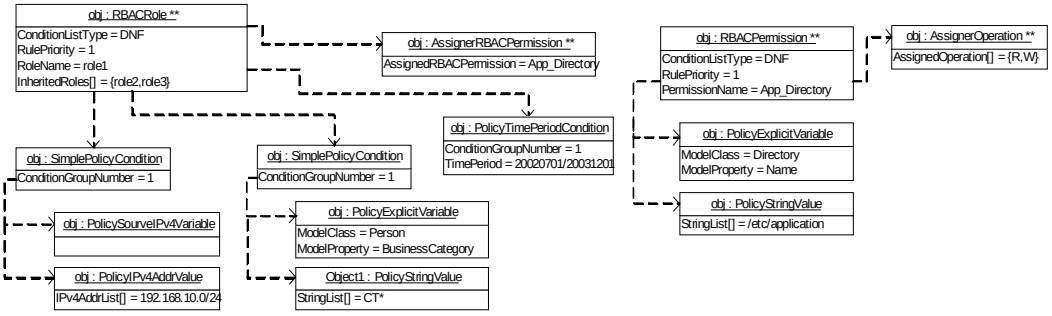


Figure 3. Object instances of the RBPIM model.

The PA relationship is defined by the reference to the permission object "*App_Directory*", shown in the Figure 3. This permission defines the operations {*R,W*} are approved when *Directory.Name* MATH "*/etc/application*". Observe how the use of explicit variables permits leveraging the information of existing CIM repositories.

5. RBPIM LDAP Mapping

As well as in PCIM, the RBPIM model is implementation neutral. RFC 3060 informs that further works will propose specific-areas mappings. In this work we propose the RBPIM mapping to LDAP according to the PCLS IETF standard [Strassner, J., 2002]. This RBPIM Schema will allow the adoption of LDAP-based directory services as RBPIM policy information repositories. Table 1 presents the RBPIM LDAP Schema main classes (Please, refer to [RBPIM 2003] for a complete list). Attributes and superior classes are not being shown. The prefix "pcim" indicates a PCIM LDAP class mapping from PCLS and the prefix "rbpim" indicates a RBPIM LDAP class mapping proposed by our work. A LDAP Schema can contain three classes types [Wahl, M., 1997]: structural, abstract and auxiliary. In this Table, respectively, (s), (ab) and (ax). A directory can only have objects (instances) from structural classes. Auxiliary classes are used to extend the attribute list permited to be used by a directory object. In this

case, a auxiliary class could be attached to a directory instance in order to allow this attribute list extension. Abstract classes are used to establish class hierarchy.

Table 1. RBPIM LDAP Schema (main classes)

LDAP Object Class	Derived from	LDAP Object Class	Derived from
rbpimRole (s)	pcimRule (ab)	rbpimSimplePolicyConditionClass (ax)	pcimConditionAuxClass(ax)
rbpimPermission (s)	pcimRule (ab)	rbpimAssignerPermissionAuxClass (ax)	pcimActionAuxClass(ax)
rbpimSSD (s)	pcimPolicy (ab)	rbpimAssignerOperationAuxClass(ax)	pcimActionAuxClass(ax)
rbpimDSD (s)	pcimPolicy (ab)	rbpimPolicyVariable (ax)	top (ab)
pcimRuleConditionAssociation (s)	pcimPolicy (ab)	rbpimPolicyValue (ax)	top (ab)
pcimRuleActionAssociation (s)	pcimPolicy (ab)	rbpimPolicyExplicitVariable (ax)	rbpimPolicyVariable (ax)
rbpimConditionAssociation (s)	pcimPolicy (ab)	rbpimPolicyImplicitVariable (ax)	rbpimPolicyVariable (ax)
pcimRuleValidityAssociation (s)	pcimPolicy (ab)	pcimTPCAuxClass (ax)	pcimConditionAuxClass(ax)

RBAC Roles and Permissions are instances, respectively, from structural classes rbpimRole and rbpimPermission. RBAC SSD and DSD constraints are from rbpimSSD and rbpimDSD classes. As defined in [Strassner, J., 2002], a policy rule can be *simple* or *complex*. In the former case, all conditions are ANDed and they can´t be grouped. In the latter case, the conditions can be grouped following the DNF/CNF semantics. The proposed RBPIM LDAP Schema uses the latter case, so rbpimRole and rbpimPermission objects group their associated conditions in the DNF/CNF semantics. For the relationship classes in PCIM, the PCLS suggests three different strategies of mappings: using LDAP auxiliary classes, using attributes representing distinguished name (DN) references, and using superior-subordinate relationships in the Directory Information Tree (DIT containment). Figure 4 presents an example containing four directory instances created through RBPIM LDAP Schema, respectively, role, permission, SSD constraint and time period constraint objects. These directory entries are in LDIF[1] format.

The directory entry *o=Bank.net (organization object class)* represents the folder of a hypothetical financial institution and *ou=Bureau1 (organizationaUnit object class)* is the entry where every policy object related to this branch is inserted. Note that the entry *o=Bank.net* has a DIT containment association with the entry *ou=Bureau1* and the entry *ou=Bureau1* has also a DIT containment association with the entry *rbpimRoleName=Accounter_I*. The entry *dn: rbpimRoleName=Accounter_I, ou= Bureau1, o=Bank.net* represents the role *Accounter_I*. Every object that represents a role is derived from the hierarchy *pcimPolicy*, *pcimRule* and *rbpimRole*. Refering to Table 1, both former object classes are abstract and the latter is a structural object class. The role *Accounter_I is enabled* (*pcimRuleEnabled* = 1) and its condition will be in the DNF semantics (*pcimRuleConditionListType* = 1). This role has two condition objects, *UsersCond1* and *UsersCond2*. Both conditions define the users whom the role *Accounter_I* will be assigned (RBAC *UA association*). This role has also a permission *Permission1* (RBAC PA *association*) and a time period *Period1* object for establishing a validity period for role activation. The RBPIM framework presented next section uses the definitions in [Howes, T., 1996] in order to construct LDAP queries to retrieve policy objects from the LDAP repository. The entry *rbpimSSDname=SSD01* creates a static separation of duty relation envolving roles *Accounter_I* and *Accounter_II*. Due to

[1] LDIF – LDAP Data Interchange Format – is a format for defining directory entries in text format. LDIF files are used by directory services for importing entries.

cardinality 2 no user can be assigned to both roles user lists. As will be shown in section 6, the RBPIM framework uses the rbpimRole's attribute *pcimRulePriority* (inherited from pcimRule) in order to select higher priorities roles until matches the specified cardinality.

dn: rbpimRoleName=Accounter_I, ou= Bureau1, o=Bank.net
objectClass: pcimPolicy
objectClass: pcimRule
objectClass: rbpimRole
rbpimRoleName: Accounter_I
pcimRuleEnabled: 1
pcimRuleConditionListType: 1
pcimRuleConditionList: pcimConditionName=UsersCond1, rbpimRoleName= Accounter_I, ou= Bureau1, o=Bank.net
pcimRuleConditionList: pcimConditionName=UsersCond2, rbpimRoleName= Accounter_I, ou= Bureau1, o=Bank.net
pcimRuleActionList: pcimActionName=Permission1, rbpimRoleName=Accounter_I, ou= Bureau1, o=Bank.net
pcimRuleValidityPeriodList: pcimValidityConditionName=Period1, rbpimRoleName= Accounter_I, ou= Bureau1, o=Bank.net
dn: rbpimPermissionName=App_Directory, ou= Bureau1, o=Bank.net
objectClass: pcimPolicy
objectClass: pcimRule
objectClass: rbpimPermission
rbpimPermissionName=App_Directory
pcimRuleConditionListType: 1
pcimRuleConditionList: pcimConditionName=Directory1, rbpimPermissionName= App_Directory, ou= Bureau1, o=Bank.net
pcimRuleActionList: pcimActionName=Operations1, rbpimPermissionName= App_Directory, ou= Bureau1, o=Bank.net
dn: rbpimSSDname=SSD01, ou= Bureau1, o=Bank.net
objectClass: pcimPolicy
objectClass: rbpimSSD
rbpimSSDname: SSD01
rbpimRoleSet: rbpimRoleName=Accounter_I, ou= Bureau1, o=Bank.net
rbpimRoleSet: rbpimRoleName=Accounter_II, ou= Bureau1, o=Bank.net
rbpimCardinality: 2
dn: pcimValidityConditionName=Period1, rbpimRoleName= Accounter_I, ou= Bureau1, o=Bank.net
objectClass: pcimRuleValidityAssociation
objectClass: pcimTPCAuxClass
pcimValidityConditionName: Period1
pcimTPCTime: 20020701T000000/20020831T240000

Figure 4. Example: RBPIM Policy Objects in LDIF format

6. RBPIM Framework

6.1. Introduction

Several IETF works describes the implementation of policy-based network management tools using the PDP/PEP approach [Yavatkar, R., 2000] [Snir, Y., 2001]. The PDP/PEP approach is a client-server model, where the PDP (Policy Decision Point) is a server responsible for supplying policy information for one or more PEP (Policy Enforcement Point) clients. Usually, the PEP is embedded in a network node responsible for enforcing the policy. For example, the PEP can be embedded in a QoS router. The IETF defines that the PEP and the PDP communicates using the COPS (Common Open Policy Service) protocol [Durham, Ed., 2000]. The COPS protocol defines two models of operation: outsourcing and provisioning. The outsourcing model assumes the PEP receives events that must be resolved based on policy criteria, e.g., the PEP is a router receiving a RSVP message asking for a reservation. In the outsourcing model, the PDP receives policy requests from a network device, and determines whether or not to grant these requests. Therefore, in the outsourcing model, the policy rules are evaluated by the PDP. By the other hand, in the provisioning model, rather than responding to PEP events, the PDP prepares and "pushes" configuration information to the PEP.

6.2. Overview

Figure 5 illustrates the main elements in the RBPIM framework. RBPIM framework adopts the PDP/PEP model using a "pure" outsourcing approach, i.e., the PDP carries most of the complexity and the PEP is comparatively light. In the RBPIM framework, the PEP is called Role-Based PEP (RBPEP). The Role-Based PDP (RBPDP) is a specialized PDP responsible for answering RBPEP questions. Observe that the RBPDP has an internal database (called State DataBase) used for storing the state information of the RBPEP. The CIM/Policy Repository is a LDAP server that stores objects that represent network information such as users, services, network nodes and policies. The Policy Management Tool is the interface for updating CIM/Policy repository information and for administrating the PDP service.

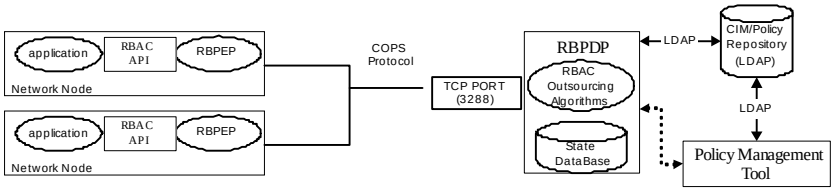


Figure 5. RBPIM Framework Overview

6.3. RBPEP and RBAC API's

The RBPEP is basically a software library that simplifies the task of building "RBAC-aware" applications. It offers a high level programming interface for mapping the RBAC APIs to COPS messages addressed to the RBPDP. The COPS is an object-oriented protocol that defines a generic message structure for supporting the exchange of policy information between a PDP and its clients (PEPs). The RBAC API's used in the RBPIM framework are based on the RBAC functional specifications described in the proposed NIST standard [Ferraiolo, D.F., 2001]. Based on the supporting system functions proposed by NIST, the RBPIM framework defines a set of five API's: *RBPEP_Open* (), *RBPEP_CreateSession(user*:string; out *session*:string, *roleset[]*:string, *usessions*:int) , *RBPEP_SelectRoles* (*session*: string, *roleset[]*:string; out *result*:BOOLEAN), *RBPEP_CheckAccess(session*: string, *operation*:string, *objectFilter[]*:string*; out result:BOOLEAN)* and *RBPEP_CloseSession*(*session*:string). The *RBPEP_Open* establishes the connection between the PEP and the PDP. The *RBPEP_CreateSession* API establishes a user session for the user and returns the set of roles assigned to the user that satisfies the SSD constraints. The user must explicitly activate the desired roles received from this call in a subsequent call called *RBPEP_SelectRoles* API. This modification avoids the need of the user to drop unnecessarily activated roles in order to satisfy DSD constraints. The *RBPEP_CheckAccess* API is similar to the standard *CheckAccess* function proposed by the NIST. The *RBPEP_CloseSession* terminates the user session, and informs to the RBPDP that the information about the session in the "state database" is no longer needed. The RBPEP_APIs are currently implemented in Java, and throws exceptions for informing the applications about the errors returned by the PDP. Examples of exceptions are: "RBPEP_client not supported", "non -existent session", "user not valid", etc. (Please refer to [Nabhen, R., 2003,1] for a complete description). These API calls are mapped to COPS messages. For example, the *RBPEP_CheckAccess* call is mapped

to the COPS REQ (Request), DEC (Decision) and RPT (Report) messages. (Please, also refer to [Nabhen, R., 2003,1] for a complete description of this COPS mapping)

6.4. The RBPDP Outsourcing Algorithms

The RBPDP module implements a set of algorithms triggered by the COPS messages sent by the RBPEPs. These algorithms interpret the RBAC policies stored in the CIM/Policy repository and the state information of the RBPEP sessions (stored in a relational state-database), and answer the RBPEP using the COPS protocol. Note that the state-database is a database internal to the *RBPDP* and its information is not described in the RBPIM model. The most important algorithms implemented by the *RBPDP* are those related to the *RBPEP_CreateSession*, *RBPEP_SelectRoles* and *RBPEP_CheckAccess.* Some obvious error treatment have been omitted in order to simplify the presentation of the algorithms. These algorithms were written based on the same notation of the NIST RBAC standard.

6.4.1. Algorithm for RBPEP_CreateSession:

The algorithm for the *RBPEP_CreateSession* API determines the set of RBAC roles assigned to the user, free of SSD constraints. Presently, the approach defined by the RBPIM framework consists in using a *RBACPolicyGroup* object for grouping the RBAC objects (this approach must be reviewed to be in conformance with the new PCIMe standard). In the CIM/Policy repository, the *RBACPolicyGroup* objects are associated to "organization units" by DIT contai nment. By using the attribute organizational unit ("OU") in the CIM *Person* object, the algorithm determines the corresponding *RBACPolicyGroup* object associated to the user. The algorithm for the *RBPEP_CreateSession* API is defined as follows:

Step 1: *If* the *session* already exists in the state database *then* returns a <*Error*> object in the DEC message. *Otherwise*, go to Step 2.

Step 2: Let *RolesInDomain*(*pg* : *RBACPolicyGroup*) ={*r* : *RBACRole* | *r* ∈ *pg***}**, be the list of role objects associated to the *RBACPolicyGroup* of the user.

Step 3: Determine *AssignedUsers*(*r*) as the list of users that satisfies the conditions associated to a role *r*. This function is implemented by creating a LDAP filter based on the conditions (*SimplePolicyCondition*) associated to the role *r* grouped in CNF or DNF form, as defined by the attribute *ConditionListType* of the role object *r*.

- *AssignedUsers*(*r*: *RBACRole*)={*u*: *cim_Person* | *u* satisfies the conditions of *r* ∈ *RolesInDomain*(*pg*)}

Step 4: Determine *AssignedRoles*(*user*) as the subset of *RolesInDomain*(*pg*) that includes only the roles *r* ∈ *RolesInDomain*(*pg*) assigned to the *user*.

- *AssignedRoles***(***user*:*cim_Person***)** ={*r* : *RBACRole* | *r* ∈ *RolesInDomain*(*pg*) ^ *user* ∈ *AssignedUsers*(*r*) }

Step 5: Determine *EnabledRoles***(***user***)** as the subset of *AssignedRoles***(***user***)** that includes only the roles that can be activated at the current time.

- *EnabledRoles***(***user* : *cim_Person***)=**{*r* : *RBACRole* | *r* ∈ *AssignedRoles***(***user***)** ^ *current time* ∈ *ActivationIntervals*(*r*)}

The function *ActivationIntervals*(*r*) returns the set of activation intervals defined by the *PolicyTimePeriodCondition* objects associated to the role *r*.

Step 6: Determine *InheritedRoles(user)*as the disjoint union of all inherited roles indicated by the attribute *InheritedRoles* of all enabled roles of the user. The InheritedRoles that can't be activated at the current time are excluded from the union.

- *InheritedRoles*(*user : cim_Person*) = $\mathbf{Y}^{*}_{r \in EnabledRoles(user)}$ *r.InheritedRoles[]* .

Step 7: Determine *AuthorizedRoles*(*user*) as the disjoint union of *EnableRoles*(*user*) and *InheritedRoles*(*user*).

- *AuthorizedRoles*(*user : cim_Person*)**=**{*r* : *RBACRole* | *r* ∈ *EnabledRoles***(***user***)** $\cup^{*}$ *InheritedRoles(user)*}

Step 8: Let *SsdRoleSets*(*pg* : *RBACPolicyGroup*)={*ssd* : *SSDRBAC* | *ssd* ∈ *pg* } be the list of *SSDRBAC* objects associated to the *RBACPolicyGroup* of the user.

Step 9: Determine *FreeSsdAuthorizedRoles(user)* by removing from *AuthorizedRoles*(*user*) the roles that are constrained by *SsdRoleSets*(*pg*). The roles with lowest priority (*RulePriority* attribute inherited by *RBACRole* from *PolicyRule*) are removed first, until the Cardinality attribute of all *SsdRoleSets*(*pg*) constraints is satisfied.

- *FreeSsdAuthorizedRoles(user : cim_Person)* ={ *r* : *RBACRole* | ∀*ssd* ∈ *SsdRoleSets*(*pg*) • |*ssd.RoleSet* ∩ *FreeSsdAuthorizedRoles(user)*| < *ssd.Cardinality* }

Step 10: Create in the state database a record with the *session*, *user*, the *roleset[]* defined by *FreeSsdAuthorizedRoles(user)* and status=*Phase1* and sends a DEC message with the parameters *roleset* and *usessions* encapsulated in <*Decision*> objects.

6.4.2. *Algorithm for RBPEP_SelectRoles:*

The *RBPEP_SelectRoles* API activate in a session the set of roles defined by the *roleset[]* argument. The *SelectRoles* API will activate the roles only if all roles in *roleset[]* are presented in the session database and all of them are free of DSD constraints. The algorithm for the *RBPEP_SelectRoles* API is defined as follows:

Step 1: *If* the *session* already exists in the state database with status=*Phase1* go to Step 2. If it doesn't, *then* returns a <*Error*> object in the DEC message.

Step 2: Let *AuthorizedSessionRoles*(*session*) be the list of role objects associated to the *session* in the state database.

- *AuthorizedSessionRoles*(*session*) ={*r* : *RBACRole* | *r* is authorized in the session}

Step 3: If *roleset*[] ⊄ *AuthorizedSessionRoles(session)* then sends a DEC message indicating the operation has been denied. Otherwise, go to Step 4.

Step 4: Let *DsdRoleSets*(*pg* : *RBACPolicyGroup*))={*dsd* : *DSDRBAC* | *dsd* ∈ *pg* } be the list of *DSDRBAC* objects associated to the *RBACPolicyGroup* of the user.

Step 5: *If roleset[]* violates the *DsdRoleSets*(*pg*) constraints then sends a DEC message indicating the operation has been denied. Otherwise, go to Step 6. The DSD constraints are vialoted if:

- ∀ *dsd* ∈ *DsdRoleSets*(*pg*) • |*dsd.RoleSet*[] ∩ *roleset*[]| ≥ *dsd.Cardinality*

Step 6: Update the state database by storing *roleset[]* as the list of active roles in the session and define status=*Phase2*. Then, sends a DEC message with *result*=true encapsulated in a <*Decision*> object.

6.4.3. *Algorithm for RBPEP_CheckAccess:*

Some considerations are necessary before presenting the algorithm for the *RBPEP_CheckAccess* API. First, remember that the *objectfilter[]* parameter in the API may contain conditions based on implicit and/or explicit variables. Explicit variable conditions may define one or more CIM objects. For example, {"*DataFile.Readable=true*", "*DataFile.Name=*.doc*"} will problably define a set of objects instead of a single object. Say **Φ** as the set of objects defined by the *objectfilter[]* in the *RBPEP_CheckAccess* API. The CIM objects in the **Φ** can be retrieved by a single LDAP query which filter is based on the *objectfilter[]* conditions. By the other hand, the *RBACPermission* objects associated to the roles activated by the user may also contain conditions based on implicit and explicit variables and, therefore, define another set of CIM objects, say **ψ**, also retrieved by a single LDAP query. The *RBPEB_CheckAccess* API will return true if **Φ** ⊆ **ψ**. Because **ψ** can be very large, the condition **Φ** ⊆ **ψ** is replaced by the equivalent expression **Φ** ⊆ **θ**, where **θ** = **ψ** ∩ **Φ**. The **θ** set can also be determined by a single LDAP query, by defining a LDAP filter that combines the conditions presented in the *objectfilter[]* and the *RBACPermission* associated conditions. The implicit variables conditions such as {"*PolicyDestinationIPv4Variable=192.168.2.3*"} are not used for creating the LDAP queries, because implicit variables doest no correspond to objects in the CIM repository. Instead, they are used for eliminating the *RBACPermission* objects that does

not satisfy the implicit variables in the *objectfilter[]* vector. The algorithm for the *RBPEP_CheckAccess* API is defined as follows:

Step 1: *Verify* if the session exists in the state database with *status=Phase2*. If it doesn't than returns an <*Error*>. Otherwise, go to Step 2.

Step 2: Let *ActiveSessionRoles*(*session*) be the list of active role objects associated to the *session* in the state database.

- *ActiveSessionRoles*(*session*) = {*r* : *RBACRole* | *r* is an active role in the session}

Step 3: Determine *EnabledSessionRoles*(*session*) as the subset of *ActiveSessionRoles*(*session*) that includes only the roles that can be activated at the current time.

- *EnabledSessionRoles*(*session*) = {*r* : *RBACRole* | *r* ∈ *ActiveSessionRoles*(*session*) ^ *current time* ∈ *ActivationIntervals*(*r*)}

The function *ActivationIntervals*(*r*) returns the set of activation intervals defined by the *PolicyTimePeriodCondition* objects associated to the role *r*.

Step 4: Determine *SessionPermissions*(*session*) as the set of permission objects corresponding to the disjoint union of the *RBACPermission* objects associated to all roles *r* ∈ *EnabledSessionRoles*(*session*). The *RBACPermission* objects associated to the role *r* are determined by the multi-valued attribute *AssignedRBACPermission[]* of *r*.

- *SessionPermissions*(*session*) = {*p* ∈ *RBACPermission* | $p \in \mathbf{Y}^{*}_{r \in EnabledSessionRoles(session)} r.AssignedRBACPermission[]$ }

Step 5: Determine *EnabledSessionPermissions*(*session*, *objectfilter[]*) as the subset of *SessionPermissions(session)* that includes only the permission objects *p* which implicit conditions are satisfied by the conditions presented in the *objectfilter*.

- *EnabledSessionPermissions*(*session*) = {*p* : *AssignerRBACPermission* | *p* ∈ *SessionPermissions(session)*^ *ImplicitObjectFilter* (*p*,*objectfilter[]*) == *true*}

Where *ImplicitObjectFilter* (*p, objectfilter[]*) evaluates the logical expression formed only by the implicit conditions of the permission object *p*, considering the implicit conditions presented in the *objectfilter[]* as *true.*

Step 6: Determine *SessionPermissionsIncludingOperation(session, operation)* as the subset of *EnabledSessionPermissions*(*session, objectfilter[]*) that includes only the permissions objects that contains the *operation* passed by the RBPEP_API.

- *SessionPermissionsIncludingOperation(session, operation)* = {*p* : *AssignerRBACPermission* | *p* ∈ *EnabledSessionPermissions*(*session, objectfilter[]*) ^ *operation* ∈ *PermissionOperations*(*p*) }

Where *PermissionOperations*(*p*) is the list of operations of the permission object *p*.

Step 7: Determine *ExplicitPermissionObjectFilter(session)* as the logical expression formed by combining the explicit conditions of all object permissions *p* ∈ *SessionPermissionsWithOperation(session, operation)* using CNF or DNF.

- *ExplicitPermissionObjectFilter(session)*= $\bigvee_{p} ExplicitPermissionObjectFilter(p)$, for ∀ *p* ∈ *SessionPermissionsIncludingOperation(session, operation)*

Where *ExplicitPermissionObjectFilter* is the logical expression formed by combining the explicit conditions of the object permission *p*.

Step 8: Determine *ExplicitObjectFilter*(*objectfilter[]*) formed by grouping the explicit conditions in the *objectfilter[]* using the AND (^) operator.

Step 9: Determine Φ = {CIM} as the list of CIM objects retrieved by the LDAP query corresponding to the expression *ExplicitObjectFilter(objectfilter[]*).

Step 10: Determine θ ={CIM}, as the list of CIM objects retrieved by the LDAP query corresponding to the expression *ExplicitPermissionObjectFilter(session)* ∧ *ExplicitObjectFilter(objectfilter[]).*

Step 11: Sends a DEC message with *result* = true if $\Phi \subseteq \theta$, otherwise, sends *result*=false.

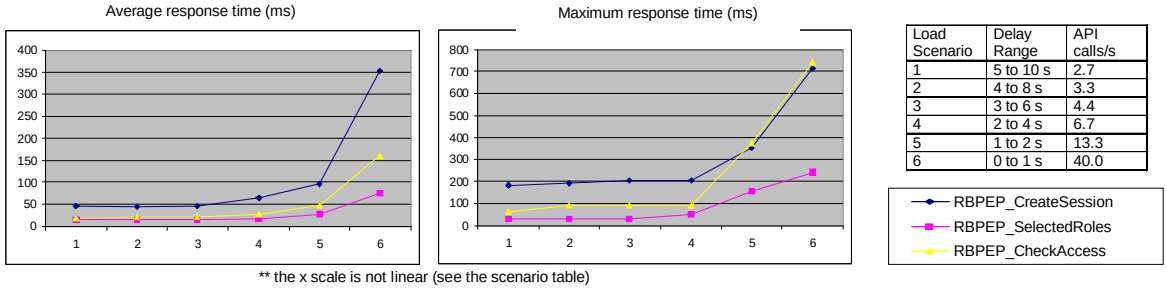
7. Evaluation

This section present the evaluation we have made based on a case study that considers a typical security policy applied in bank bureaus. The security policy takes into account individuals, positions, authorization schemes, activities and privileges used in the organization. The Bank Bureau uses a great number of applications to support its business procedures. The RBPIM will be used for establishing access policies in order

to control the access to each operation provided by those applications.

In order to evaluate the performance the RBPIM framework, a Java based RPPDP and a RBPEP scenario simulator was implemented. This prototype is available for download in [RBPIM 2003]. In the evaluation scenario, twenty RBPEP clients request the RBPIM policy service provided by a single PDP. Each RBPEP keeps a distinct COPS/TCP connection with the PDP. The RBPEP clients simulate typical access control scenarios created by text input files. Each line of these input files corresponds to an API call presented in section 6.3. Several user sessions were created in the context of each RBPEP connection. For each connection served, the RBPDP generates an output file containing all COPS messages associated with the correspondent API call in the input file and the elapsed time from the instant of receiving the RBPEP's COPS message to the PDP's decision. In order to simulate different load scenarios, we have introduced a uniformly distributed random delay between each API call contained in the input files. By varying the range of the random delay, we have created six load scenarios as shown in Figure 6. The load scenario "1" is the lightest scenario and the number "6" is the heaviest one. The former makes the PDP to receive 2.7 requests/second (average) and the latter increases this number to 40 requests/second (average). The Figure 6 presents the results obtained with the Java prototype, using a Pentium IV 1.5 Ghz 256 Mb RAM for hosting the PDP, and other identical machine for hosting the 20 RBPEP clients.

The results of the evaluation tests show the number of role objects as the most important parameter affecting the response time in the RBPIM framework. The results also show reasonable response times considering the Java implementation and the CPU capacity of the machines used in the simulation. A response time of 50 ms for *RBPEP_CreateSession* (100 ms with twenty roles) in scenario 4 is a reasonable result for an API that is evocated only once in a session. Also, the *RBPEP_CheckAccess* average response time API has presented reasonable results for applications that requires decisions based on user events, and is not significantly affected by the number RBAC policy objects.



Load Scenario	Delay Range	API calls/s
1	5 to 10 s	2.7
2	4 to 8 s	3.3
3	3 to 6 s	4.4
4	2 to 4 s	6.7
5	1 to 2 s	13.3
6	0 to 1 s	40.0

** the x scale is not linear (see the scenario table)

Figure 6. RBPDP decision time x API calls.

8. Conclusion

This paper has presented a complete policy based framework for implementing RBAC policies in heterogeneous and distributed systems. This framework, called RBPIM, has been implementing in accordance with the IETF standards PCIM and COPS, and also, the proposed NIST RBAC standard. The framework proposes a flexible RBAC model, which permits specifying the relationship between users, roles, permissions and objects by combining Boolean expressions. The performance evaluation of the outsourcing

model indicates that this approach is suitable for supporting RBAC applications that requires decisions based on user events. This paper does not discuss the problems that could rise if the PDP breaks. Future works must evaluate alternative solutions for introducing redundancy in the PDP service. These studies will be carried out in parallel with the evaluation of provisioning and hybrid approaches for implementing the RBPIM framework. Also, some important PCIMe modifications must be taken into account in a revised version of the RBPIM information model. Finally, some studies are being developed for evaluating the use of the RBPIM framework for QoS management based on RBAC rules.

9. References

[Bartz, L. 1997] "LDAP Schema for Role Based Access Control", IETF Internet Draft, October.

[Bartz, L. 2001] "CADS-2 Information Model", not published, IRS: Internal Revenue Service.

[Distributed Management Task Force 1999] "Common Information Model (CIM) Specification", URL: http://www.dmtf.org.

[Distributed Management Task Force 2000] "Guidelines for CIM-to-LDAP Directory Mappings", whitepaper, May 8th, URL: http://www.dmtf.org.

[Durham, Ed.,Boyle, J., Cohen, R.,Herzog, S., Rajan, R., Sastry A. 2000] The COPS (Common Open Policy Service) Protocol, IETF RFC 2748, January.

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

[Howes, T., Smith, M. 1996] A LDAP URL Format. Request For Comments 1959, June.

[Moore, B., Elleson, E., Strasser, J., Westerinen, A. 2003] "Policy Core Information Model Extensions", IETF RFC 3460, January.

[Moore, B.,Elleson, E., Strasser, J., Westerinen, A. 2001] "Policy Core Information Model", IETF RFC 3060, February.

[Nabhen, R., Jamhour, E., Maziero C. 2003,1] "RBPIM: A PCIM-Based Framework for RBAC", Proceedings for the 28th Annual IEEE International Conference on Local Computer Networks, October, Germany.

[Nabhen, R., Jamhour, E., Maziero C. 2003,2] "Policy-Based Framework for RBAC", Proceedings for the fourteenth IFIP/IEEE International Workshop on Distributed Systems: Operations & Management, October, Germany.

[Nabhen, R., Jamhour, E., Maziero C. 2003,3] "A Policy Based Framework for Access Control", Proceedings for the Fifth International Conference on Information and Communications Security, October, China.

[OASIS 2003] "eXtensible Access Control Markup Language (XACML) –Version 1.03", OASIS Standard, 18 February, URL: http://www.oasis-open.org.

[RBPIM 2003] Project WebSite. URL: http://www.ppgia.pucpr.br/~jamhour/RBPIM, April.

[Snir, Y.,Ramberg, Y.,Strassner, J.,Cohen, R.,Moore B. 2001] "Policy QoS Information Model", IETF internet-draft, November.

[Strassner, J.,Ellesson, E.,Moore, R. 2002] "Policy Core LDAP Schema", Internet Draft, January.

[Wahl, M., Coulbeck, A., Howes, T. , Kille, S. 1997] "Lightweight Directory Access Protocol (v3): Attribute Syntax Definitions", RFC 2252, December.

[Yavatkar, R.,Pendarakis, D.,Guerin R. 2000] "A Framework for Policy-based Admission Control", IETF RFC 2753, January.

[Yeong, W.,Howes, T., Killie, S. 1995] "LightWeight Directory Access Protocol", RFC 1777, March.