Role-Based Access Control for CORBA Distributed Object Systems

Rafael R. Obelheiro and Joni S. Fraga
Email: rro@das.ufsc.br, fraga@das.ufsc.br
Department of Automation and Systems
Federal University of Santa Catarina
C. P. 476 – 88040-900 – Florianópolis – SC – Brazil

Abstract

This paper shows how role-based access control (RBAC) models can be implemented in distributed object-based systems that follow OMG/CORBA standards. We introduce a novel approach that provides for automatic role activation by the security components of the middleware, which brings role-based access control to security-unaware applications.
Key words*: security, access control, RBAC, CORBA*

1. Introduction

Organizations have become increasingly dependent on their information systems. As such, they are also becoming very concerned with ensuring security of the information managed and stored by these systems. The observed growth in deployment of large-scale distributed systems, especially those based on the Internet, brings new challenges to the task of maintaining data confidentiality, integrity and availability. This stems from several factors, including the impossibility of providing physical security to all components of the system and also the ease of remote access, which turn geographical boundaries and regulatory laws (concerning access and use of this information) irrelevant.

At the same time, we notice the dissemination of information systems based on Common Object Request Broker Architecture (CORBA)[1] technology, a standard for open distributed object-based systems that has gained worldwide adoption by the software industry [2].

Among several services specified by OMG for the CORBA environment is the CORBA Security Service, also called CORBASec. The CORBA security model was designed to incorporate security features into both small-scale and large-scale distributed object systems without leaving aside important advantages of CORBA such as transparency, interoperability and portability.

Role-based access control has been being recognized as an alternative to traditional discretionary (based on an access matrix) and mandatory (based on security labels) access control models. Studies by the US National Institute of Standards and Technology (NIST) in the beginning of the 1990s [6] show that many organizations want the access to information to be controlled according to a centralized policy in a flexible way, so that it easily adapts to new requirements that arise naturally as the organization evolves. Clearly these objectives are difficult to achieve simultaneously using either discretionary (flexible but decentralized) or mandatory (centralized but inflexible) access controls. Role-based access control, on the other hand, can satisfy both requirements effectively [11].

This work shows how to integrate role-based access control into open distributed systems that follow OMG/CORBA standards. We introduce a novel strategy that consists of automatic role activation by the security components of the middleware. Existing proposals for implementing RBAC require that users or applications interact with the security subsystem to select which roles should be activated in the system. Our approach, however, allows users and applications to be isolated from these details so that security policies can be deployed without changing existing applications and without modifying the way users interact with these applications.

This paper is organized as follows. Section 2 describes the RBAC model this work is based on. Section 3 presents the CORBA security model. Section 4 discusses how RBAC can be implemented within a CORBASec framework, and section 5 presents some results obtained with an implementation prototype. Finally, section 6 discusses related work and section 7 presents our conclusions.

[1] The Object Management Group (OMG), a consortium formed by more than 800 companies, is the organization responsible for the specification of the CORBA architecture standards.

Proceedings of the Seventh International Workshop on Object-Oriented Real-Time Dependable Systems (WORDS 2002)
1530-1443/02 $17.00 © 2002 **IEEE**

2. Role-Based Access Control—RBAC

Role-based access control (RBAC) as a concept has arisen with the first interactive multi-user computing systems, in the early 1970s. The central idea behind RBAC is that users are assigned to roles, permissions are assigned to roles and users acquire permissions by being members of roles. Roles are created according to the different jobs or functions within an organization, and users are assigned to roles according to their responsibilities and qualifications. Users can be easily reassigned from one role to another. Changes in the computing environment, such as installation of new systems and removal of old applications, modify only the set of permissions assigned to the different roles, without directly involving the set of users.

The reference model used in this work is shown in figure 1. This model corresponds to the Symmetric RBAC model from the unified NIST RBAC family of models [12]. It has four sets of entities: users (*U*), roles (*R*), permissions (*P*) and sessions (*S*). A user in this model is a human being or a process acting on his behalf. A role is a function or job within an organization with some associated semantics regarding the authority and responsibility conferred on a member of this role. A permission is an approval of a particular mode of access to one or more objects in the system. The unified NIST RBAC model does not define specific permissions; this is left to implementors of the model [12]. A session corresponds to a user accessing the system with a given set of active roles.

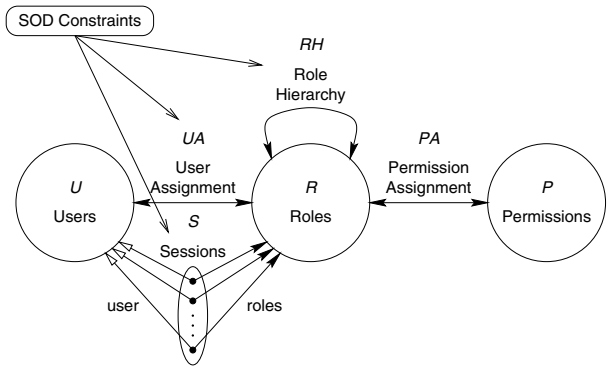

Figure 1. Symmetric RBAC reference model

The user-role assignment (*UA*) and the permission-role assignment (*PA*) are many-to-many relations. Set *U* of users has a one-to-many relation with set *S* of sessions (a user can have many sessions), and set *S* has a many-to-many relation with set *R* of roles (a session can have many roles, and a role can be active in many sessions).[2]

[2]Many-to-many relations are represented in figure 1 by filled arrowheads, while one-to-many relations are represented by empty arrowheads.

Role hierarchies, represented by relation *RH* in figure 1, are a natural means for structuring roles to reflect an organization's lines of authority and responsibility. These hierarchies are mathematically represented by partial order relations [11].

The RBAC model shown in figure 1 also supports the concept of **separation of duty** (SOD), which is an important technique for minimizing the occurrence of error and fraud in information handling. This principle consists of dividing individual operations in several smaller subtasks which should be carried out by different people, thus reducing the individual power of each user. Separation of duty has had its importance for information security recognized and discussed in detail by Clark and Wilson [4]. RBAC enforces separation of duty by means of mutual exclusion of roles.

There are two basic forms of separation of duty, static and dynamic. In **static separation of duty** (SSD), two roles R1 and R2 which are mutually exclusive cannot have users in common; in other words, the same user cannot be assigned simultaneously to both R1 and R2. On the other hand, in **dynamic separation of duty** (DSD) a mutual exclusion between two roles R1 and R2 means that a user can be assigned to both roles, provided that only one of them (either R1 or R2) is active at a given moment [12].

3. CORBA Security Model

The Object Management Group developed a security reference model for CORBA distributed object systems [10]. The CORBA Security specification defines a set of objects and their relations in a model that provides functionalities such as principal identification and authentication, access control, secure communication between objects, nonrepudiation, audit and security management.

According to the CORBASec specification, a secure distributed object system can be divided in four levels. The application level contains application objects (clients and servers). The middleware level comprises ORB services, object services (COSS—Common Object Service Specification) and the ORB core. ORB services and COSS object services are built upon the ORB core and extend the basic functions with additional features, implementing security at the middleware level. The security technology level corresponds to the underlying security services, which define protocols used for guaranteeing properties such as confidentiality and integrity in client-server communication. The lower level is the basic protection level, which comprises the underlying operating system and hardware protection.

The CORBASec specification defines a set of object services that implement the security controls in a secure CORBA system. These are PrincipalAuthenticator, Credentials, AccessPolicy, RequiredRights, AccessDecision, Secu-

rityManager, Current, PolicyCurrent, Vault and SecurityContext.

The CORBA security model uses the concept of **principal** to mediate method invocations in the system. A principal is a user or system entity that is registered in, and authentic to, the distributed object system [10]. The PrincipalAuthenticator object is responsible for authentication of principals in the CORBASec architecture. This authentication enables a principal to obtain its credentials.

The **credential** of a principal contains its identity and privilege attributes; these attributes enable the principal to perform invocations to server objects in the system. In a secure CORBA system, a credential is represented by a Credentials object. Usually there is one credential per security mechanism used in a secure CORBA system. This approach is used, for instance, in ORBAsec SL2, a secure ORB that implements the CORBASec specification [1]. ORBAsec SL2 uses Kerberos and SSL (Secure Sockets Layer) as security mechanisms, using a different credential for each mechanism.

Security policies are expressed in terms of security attributes of system resources (control attributes) and of principals (privilege attributes). Authorization policies are represented in the CORBASec specification by an AccessPolicy object; an example is illustrated in figure 2. This object contains the rights[3] granted to principals for invoking operations in a secure CORBA system, based on their privilege attributes. Only the rights `g` (get), `s` (set), `m` (manage) e `u` (use)—which belong to the predefined family `corba`—are defined by the CORBASec specification, although it is possible to freely define other rights families and types.

Privilege Attribute	Granted Rights
`role: client`	`corba: gs--`
`role: client`	`corba: g---`
`role: manager`	`corba: g-mu`
`role: manager`	`corba: g--u`

Figure 2. AccessPolicy **example**

The rights required for execution of operations in server objects (the control attributes) are stored in a RequiredRights object. As figure 3 shows, required rights are specified per interface (a class according to OMA) and not per instance (an individual object). In addition, there is also a combinator, which indicates if a principal needs to have *All* required rights to invoke an operation or if having *Any* of them is sufficient.

The AccessDecision object is responsible for deciding if an invocation of an operation of a given server should be allowed or not. This access decision depends on privilege attributes (represented by the AccessPolicy object) and on control attributes (represented by RequiredRights). The logic of this access decision is left open by the CORBASec specification [2].

[3]The concept of **rights** used in the CORBA security model is equivalent to the concept of **permissions** used in the RBAC model. Both terms are used indistinctly in this work.

Required Rights	Combinator	Operation	Interface
`corba: g---`	All	`get_balance`	`PersAcc`
`corba: g---`	All	`get_balance`	`CorpAcc`
`corba: -sm-`	Any	`open`	`PersAcc`
`corba: g-m-`	All	`open`	`CorpAcc`

Figure 3. RequiredRights **example**

Session objects—SecurityManager, PolicyCurrent and Current—store information about the current security context, such as references to the RequiredRights and AccessDecision objects (stored in SecurityManager), references to policy objects used to establish secure associations (PolicyCurrent) and the principal's credentials obtained by the server (Current). The Vault and SecurityContext objects take part in the establishment of **secure associations**, which guarantee confidentiality and/or integrity of messages exchanged between client and server.

The CORBASec specification defines two kinds of interceptors[4] which are called during an operation invocation. The first is the **access control interceptor**, a high-level interceptor which performs access control functions, and the second is the **secure invocation interceptor**, which is a low-level interceptor that provides confidentality and integrity to the messages exchanged between client and server. These interceptors are created when the client binds to the server, and perform different functions at distinct moments of a method invocation. At bind time, the control access interceptor is responsible for instantiating the AccessDecision object and updating its reference in the SecurityManager object. The AccessDecision object should provide the domain-specific AccessPolicy policy object, inserting its reference in the PolicyCurrent object, and also locate the RequiredRights object, updating its reference in the SecurityManager object. At access decision time, the access control interceptor invokes the `access_allowed` operation of the AccessDecision object, which is responsible for authorizing or denying the method invocation, obtaining the granted rights from AccessPolicy and comparing them to the required rights for the given operation (according to RequiredRights).

4. The RBAC-JACOWEB Proposal

The JACOWEB project, developed at LCMI–DAS–UFSC, aims to investigate the problem of authorization in

[4]In the CORBA security model, ORB services are implemented by means of **interceptors**, which are objects logically interposed in the sequence of an invocation between a client and a server. Each security-related COSS service is associated to an interceptor, which is transparently inserted into the invocation path to activate the associated service.

large-scale distributed systems. The project is focused on defining and implementing authorization schemes for distributed applications, using, as a foundation, the CORBA security model integrated with the Java and Web security models [14]. In the JACOWEB architecture, access control is performed by the middleware, in a transparent manner, in both sides (client and server) of an invocation. Object services in the client side verify a request against the defined security policy, checking if it should or not be authorized. If the access is authorized, a capability is generated and added to the CORBA request to be sent to the server; object services in the server side extract and validate the capability before effectively granting access.

This paper introduces the RBAC-JACOWEB proposal, whose goal is to incorporate a role-based access control model into the JACOWEB authorization scheme. The RBAC model used is the Symmetric RBAC from the unified NIST RBAC model, presented in section 2. The idea here is to extend the PoliCap policy service defined in [15] so that this service becomes responsible for managing the RBAC configuration in a CORBASec context.

4.1. The PoliCap Policy Service

PoliCap is a policy service for distributed objects whose invocations are regulated according to the CORBA security model [15]. It has been developed in the context of project JACOWEB and corresponds to a first level of access control in the authorization scheme.

PoliCap allows for centralized management of policy objects within a distributed objects security domain, filling a void in the CORBASec specification regarding management of policy objects. According to the specification, security policies are made available by the AccessPolicy and RequiredRights objects. With PoliCap, administrative applications interact with the policy service in order to manage these objects. On the other hand, operational applications or COSS services interact with PoliCap to obtain, at bind time, the policies and rights needed to control method invocations at run time. The idea here is that, at bind time, the policy service provides local versions of the AccessPolicy and RequiredRights objects that contain the privilege and control attributes which are appropriate to the given invocation. Access decisions are then performed based on these local instances of AccessPolicy and RequiredRights. Further detail on PoliCap can be found in [15].

4.2. Integrating RBAC Models with CORBASec

In the RBAC-JACOWEB proposal, each principal has only one credential (since SSL is the only security technology supported by JACOWEB). The roles assigned to a principal are represented, in his credential, by privilege attributes of type Role. After a principal is authenticated, his credential contains only his AccessId, which is a privilege attribute that represents the principal's identity for access control purposes, and that can be extracted from his SSL certificate (used for establishing a secure association). Roles are added to the credential as they are activated in the system. Therefore, a principal's credential represents, in our proposal, his access identification and his set of active roles.

Security interceptors remain with the same functionality described in section 3. The modifications to incorporate the role-based access control are made to the AccessDecision object, which verifies if the rights granted to the principal allow him to invoke a given operation. As an invocation is authorized, the client access control interceptor generates a capability which will be added to the CORBA request.

PoliCap contains all data concerning security policies within a domain, including users, roles, user-role and role-permission assignments, role hierarchy relations and separation of duty constraints. The role-based access control is performed by the operation `role_access` of PoliCap, whose IDL interface is shown in figure 4.

```
boolean role_access
  (inout SecurityLevel2::CredentialsList cred_list,
   in CORBA::Identifier operation_name,
   in CORBA::Identifier target_interface_name,
   inout SecurityAdmin::AccessPolicy local_ap);
```

Figure 4. IDL interface for operation `role_access` of PoliCap

Based on the client's credentials and on the invoked operation, PoliCap decides if the access should be authorized or not. If the user is assigned to one or more roles that grant him the permissions needed to perform the invocation and if these roles can be activated (that is, such activation does not violate any constraints), then access is granted, and `role_access` returns `true`. If it is not possible to activate roles that grant the necessary permissions, the operation returns `false`.

When access is authorized, the principal's credentials are augmented with the newly activated roles, and argument `local_ap` (which represents the local AccessPolicy object) is modified to incorporate the new rights granted by these roles.

Static separation of duty constraints are verified by the operation of PoliCap that assigns users to roles: a user cannot be assigned to a role that has a static SOD constraint with another one which he is already assigned to. On the other hand, dynamic SOD constraints are handled by the `role_access` operation. A role can only be activated if there are no dynamic constraints between such role and any of the active roles (contained in the user's credential). Do-

ing this we ensure that an automatic role activation will not violate the defined security policy.

4.3. The Dynamics of Role-Based Access Control

4.3.1. Binding. The binding happens whenever a client makes an invocation to a server he has not invoked before. In the first place, the access control interceptor uses the `role_access` operation to perform a global policy check, that is, PoliCap verifies if the principal's rights are sufficient for her to invoke the designated operation. If the rights are insufficient, access is denied, the invocation is interrupted with an exception being returned to the client and the event should be recorded by the CORBASec audit service. If, on the other hand, her rights are sufficient, her credentials and the local AccessPolicy object are updated, and the binding proceeds with the interceptor retrieving the required rights for the target interface and subsequently updating the local RequiredRights object. Furthermore, as discussed in section 3, the AccessDecision object is instantiated and its reference is updated in SecurityManager.

4.3.2. Access Decision. At access decision time, the access control interceptor in the client side calls operation `role_access` of AccessDecision, which checks if the rights granted to the principal allow her to invoke the desired method. If the access is authorized, the invocation sequence proceeds normally, with a capability being generated and sent to the server.

On the other hand, if the granted rights are not sufficient, AccessDecision invokes `role_access` so that the policy service can verify if it is possible to activate new roles that grant the rights needed to perform the invocation desired. If access is granted with the activation of one or more roles, the access control interceptor should then generate a capability and proceed with the invocation. If the RBAC configuration does not allow the access, the invocation is terminated with denial of access and the event should be recorded by the audit service.

4.4. Example

We shall consider an example of the proposed role-based access control in a banking application. The set of roles in the system is $R = \{\mathsf{cust}, \mathsf{cpers}, \mathsf{ccorp}, \mathsf{man}\}$, representing, respectively, bank customers, clerks for personal accounts, clerks for corporate accounts and managers. RBAC configuration is shown in figure 5. There are no static separation of duty constraints. Dynamic separation of duty constraints are given by figure 5(a), where $(\surd)$ denotes that the line and column represent mutually exclusive roles. Figure 5(b) represents the AccessPolicy object for the domain. Figure 5(c) shows the user-role assignment (set *UA*). The RequiredRights object is represented by figure 5(d).

Figure 6 shows an example scenario for the proposed role-based access control, focusing on the evolution of the set of active roles in the system. In this example, principal `bob` invokes the operations in the first column in the order shown by the figure. Columns *AR Before* and *AR After* represent the set of active roles—stored in the client's credential—before and after the access decision for the corresponding operation.

Next, we show in details how the CORBA security model objects and PoliCap interact to enforce access control, using the scenario in figure 6 as a basis.

4.4.1. Execution of the scenario in figure 6. When the secure CORBA system is initialized, principal `bob` is authenticated through PrincipalAuthenticator. His credential (represented by the Credentials object) contains only his AccessId, extracted from his SSL certificate:

Attribute Type	Attribute Value
AccessId	`bob`

▷ **Invocation of operation `PersAcc::open`:**

To invoke this operation, the client has to bind itself to the PersAcc server object. At this moment, PoliCap is used to retrieve the required rights for interface PersAcc and the rights that are granted to principal `bob` and which authorize him to perform the `open` operation. This is also when AccessDecision is instantiated and its reference updated on SecurityManager.

The permissions needed to invoke `PersAcc::open` are `corba: -s--` or `corba: --m-`, as the combinator is *Any*. In this case, role cpers is activated during the binding, since it gives the permissions `corba: gs--`, and the access is authorized. The principal's Credentials object now becomes the following:

Attribute Type	Attribute Value
AccessId	`bob`
Role	cpers

The local version of AccessPolicy contains the data below:

Privilege Attribute	Granted Rights
`role: cpers`	`corba: gs--`

The local RequiredRights object, after the binding, is constituted by:

Required Rights	Combinator	Operation	Interface
`corba: g---`	All	`get_balance`	`PersAcc`
`corba: -s--`	All	`deposit`	`PersAcc`
`corba: -sm-`	Any	`open`	`PersAcc`

	cust	cpers	ccorp	man
cust	—	√	√	√
cpers	√	—		√
ccorp	√		—	√
man	√	√	√	—

(a) Dynamic SOD

Privilege Attributes	Granted Rights
role: cust	corba: g---
role: cpers	corba: gs--
role: ccorp	corba: g--u
role: man	corba: g-m-

(b) AccessPolicy

Principal	Roles
anna	cust, ccorp, man
bob	cust, cpers, ccorp
chris	cust, cpers

(c) Set *UA*

Required Rights	Combinator	Operation	Interface
corba: g---	All	get_balance	PersAcc
corba: g---	All	get_balance	CorpAcc
corba: -s--	All	deposit	PersAcc
corba: ---u	All	deposit	CorpAcc
corba: -sm-	Any	open	PersAcc
corba: g-m-	All	open	CorpAcc

(d) RequiredRights

Figure 5. RBAC configuration managed by PoliCap

Operation	*AR* Before	*AR* After	Notes
PersAcc::open	∅	{cpers}	This operation requires one of the rights -sm-, and role cpers grants the rights gs--. Access is granted, with activation of role cpers.
PersAcc::deposit	{cpers}	{cpers}	This operation requires the right -s--, already conferred by role cpers. Access granted.
CorpAcc::deposit	{cpers}	{cpers, ccorp}	This operation requires right ---u, granted by role ccorp. Access granted, with activation of role ccorp.
CorpAcc::open	{cpers, ccorp}	{cpers, ccorp}	This operation requires the rights g-m-, but --m- is not granted by any role assigned to principal bob. **Access denied.**

Figure 6. Example scenario for principal bob

▷ **Invocation of operation `PersAcc::deposit`**:

As the client is already bound to PersAcc, the checks regarding PersAcc::deposit are limited to the access decision procedure shown in section 4.3.2. For this purpose, the access control interceptor invokes AccessDecision::access_allowed. The deposit operation requires the right corba: -s--, already in the local AccessPolicy object. Thus, the access is authorized, without changing Credentials and the local AccessPolicy.

▷ **Invocation of operation `CorpAcc::deposit`**:

For this invocation it is necessary another binding, this time between the client and CorpAcc. Once again, the required rights (this time for interface CorpAcc) and the granted rights which authorize the invocation of deposit are obtained from PoliCap.

The right needed to invoke CorpAcc::deposit is corba: ---u, conferred by role ccorp, which is activated. The access is authorized and object Credentials is modified, and its new contents are the following:

Attribute Type	Attribute Value
AccessId	bob
Role	cpers
Role	ccorp

The local AccessPolicy object is also updated:

Privilege Attribute	Granted Rights
role: cpers	corba: gs--
role: ccorp	corba: ---u

The local RequiredRights is now constituted by:

Required Rights	Combinator	Operation	Interface
corba: g---	All	get_balance	PersAcc
corba: -s--	All	deposit	PersAcc
corba: -sm-	Any	open	PersAcc
corba: g---	All	get_balance	CorpAcc
corba: ---u	All	deposit	CorpAcc
corba: g-m-	All	open	CorpAcc

▷ **Invocation of operation `CorpAcc::open`:**

The client is already bound to server CorpAcc, so the access control verification goes straight to the access decision procedure. Operation `CorpAcc::open` requires the rights `corba: g-m-`, which are not present in the local AccessPolicy object. Therefore, operation `AccessDecision::access_allowed` invokes the `role_access` operation to verify if the required rights can be granted by a role activation. However, principal `bob` is not assigned to any role that grants the necessary rights, so access is denied, without modifications neither in the client's credentials nor in the local AccessPolicy object.

It should be noticed that the local versions of AccessPolicy and RequiredRights are accessed through references stored in the SecurityManager session object; these references are valid for all bindings established between a client and one or more servers during a session (that is, while the client is active in the system). This explains the fact that AccessPolicy and RequiredRights are simply updated (and not re-instantiated) as the system evolves.

5. Implementation Results

A prototype of the RBAC-JACOWEB proposal has been developed in our laboratories. To test and refine this prototype, we used a banking application as an example. This application comprises two CORBA server objects and a Java client applet. The CORBA servers were developed with JacORB [3], a free Java ORB, and the applet was developed with Java 2 SDK, version 1.2.1. The browser used for testing was Netscape Communicator 4.76 with Java plug-in. The goal of this implementation was to determine the effectiveness of the proposed role-based access control scheme. The same system architecture had been used previously to successfully implement a discretionary authorization scheme [15]. The structures already developed were reviewed and adapted, serving as a basis for the new role-based authorization scheme.

The key components of the prototype are the access control interceptors presented in section 3 and a version of the PoliCap policy service augmented with RBAC support (as described in section 4.2). A version of the AccessDecision object which interacts with PoliCap, which is central to the RBAC-JACOWEB proposal, was developed. The client's credentials—containing only an AccessId privilege attribute—are generated as the secure system is initialized, based on the client's SSL certificate. A graphical interface for managing the RBAC configuration has also been implemented. It gives a security administrator the ability to manage users, roles and permissions as well as role hierarchies and separation of duty constraints.

The scenario shown in figure 6 was used as a test case. The server objects PersAcc and CorpAcc were implemented, as well as an applet that repeats the sequence of operations shown in that figure. The role-based authorization scheme implemented was able to cope with this and other scenarios, demonstrating the viability and adequacy of the RBAC-JACOWEB proposal.

The current version of the prototype does not perform authentication of principals through PrincipalAuthenticator yet. The client's credential is initialized with the AccessId extracted from her SSL certificate, but this client does not go through a process of authentication. There are enhancements to be made to the management interface, such as presenting role hierarchies in a graphical manner. The role selection mechanism embedded in the `role_access` operation is not as sophisticated as it can be; the algorithm is being refined to make better choices as for which roles should be activated for a given operation, with the objective of minimizing the permissions acquired by the user by means of a role activation.

6. Related Work

Although the basic concept of role has been used for decades as a mechanism for permission management, formal RBAC models have only recently arisen. The first RBAC model formalized in literature is due to Ferraiolo and Kuhn [7], both from NIST. Sandhu's work [11] was the first one to recognize the impossibility of capturing all the nuances of RBAC in a single model, which led to the definition of the RBAC96 family of models. The original NIST model was subsequently revised, and some of its concepts have been updated over time [5]. All these models share a common core of concepts, but each one has its own peculiarities which make them different. There are, however, visibly more similarities than differences. This fact led NIST and the group headed by Sandhu to propose a unified model that standardized RBAC concepts, in an attempt to establish consensus and also to serve as a starting point for new developments. This model, largely based on the NIST model and on the RBAC96 family, is known as the unified NIST RBAC model [12]. Even though some aspects of the unified model have been contested [9], the proposal was well received by the RBAC community, and the model is currently undergoing revision.

RBAC/Web [5] is an intranet application where RBAC is used as an authorization scheme to control access to pages in a Web server. The RBAC model used as reference is the NIST model. Users in RBAC/Web correspond to user logins in the Web servers, and the HTTP transactions that can be performed by users (through their roles) on RBAC-protected pages represent permissions. The user is responsible for choosing, among the roles assigned to him, which ones should be activated during a session. SSL is not part

of the application itself, but it is considered to be a part of its operational context.

The JRBAC-Web proposal [8] also concerns Web server security, but it differs from RBAC/Web for being focused on the Java security model. RBAC is implemented as a Java Authorization and Authentication Service (JAAS) extension, and it is based on its own reference model. As in RBAC/Web, permissions are represented by HTTP transactions, but there is no connection between RBAC users and external entities (such as a user login). In this proposal, applications (which are Java servlets) are responsible for role activation. The main advantages of the RBAC-JACOWEB proposal in comparison with these two experiences are its greater flexibility (RBAC-JACOWEB can be used with any CORBA application), its transparency (roles are automatically activated by the system) andk its adherence to standards such as the unified NIST RBAC model and the CORBA Security specification.

Beznosov and Deng [2] define a framework for implementing RBAC using the CORBA Security service. The main differences between our proposal and this framework are transparency to applications and users and the RBAC model used as reference. In Beznosov and Deng's proposal, the user interacts with a PrincipalAuthenticator object (through a UserSponsor) to select the active roles in a session, and the RBAC model used is the RBAC96 family [11]. On the other hand, in the RBAC-JACOWEB proposal roles are automatically activated by PoliCap as needed, in a transparent manner, and the reference RBAC model used is the unified NIST model [12].

7. Conclusions

This paper presented the RBAC-JACOWEB proposal that shows how role-based access control can be integrated into CORBA distributed object-based systems using standards such as the CORBA Security specification and the unified NIST RBAC model. Our main contribution, however, is the introduction of automatic role activation by the security subsystem, an innovative approach according to the known literature about RBAC.

The prototype developed within project JACOWEB shows the effectiveness of our proposal and also of RBAC as an access control model that is both flexible (in the definition of security policies) and rigorous (in the enforcement of defined policies).

As RBAC-JACOWEB evolves, we intend to add an administrative RBAC model such as ARBAC99 [13] to our framework. The graphical management interface for PoliCap will be enhanced and extended with new features that might become useful.

References

[1] Adiron. *ORBAsec SL2 User Guide, version 2.1.4*. Syracuse, NY, July 2000.

[2] K. Beznosov and Y. Deng. A Framework for Implementing Role-Based Access Control Using CORBA Security Service. In *Proceedings of the 4th ACM Workshop on Role-Based Access Control (RBAC'99)*, pages 19–30, Fairfax, VA, 1999.

[3] G. Brose. JacORB—Design and Implementation of a Java ORB. In *Proc. DAIS'97*, pages 143–154, Sept. 1997.

[4] D. Clark and D. Wilson. A Comparison of Commercial and Military Computer Security Policies. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 184–194, Oakland, CA, 1987.

[5] D. F. Ferraiolo, J. F. Barkley, and D. R. Kuhn. A Role-Based Access Control Model and Reference Implementation Within a Corporate Intranet. *ACM Transactions on Information and Systems Security*, 2(1):34–64, Feb. 1999.

[6] D. F. Ferraiolo, D. M. Gilbert, and N. Lynch. Assessing Federal and Commercial Information Security Needs. NISTIR 4976, National Institute of Standards and Technology, Gaithersburg, MD, Nov. 1992.

[7] D. F. Ferraiolo and D. R. Kuhn. Role-Based Access Controls. In *Proceedings of the 15th NIST-NCSC National Computer Security Conference*, pages 554–563, Baltimore, MD, 1992.

[8] L. Giuri. Role-Based Access Control on the Web Using Java. In *Proceedings of the 4th ACM Workshop on Role-Based Access Control (RBAC'99)*, pages 11–18, Fairfax, VA, 1999.

[9] T. Jaeger. Rebuttal to the NIST RBAC Model Proposal. In *Proceedings of the 5th ACM Workshop on Role-Based Access Control (RBAC'2000)*, pages 65–66, Berlin, Germany, 2000.

[10] Object Management Group. Security Service Specification, version 1.7. OMG Document 99-12-02, Dec. 1999.

[11] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman. Role-Based Access Control Models. *IEEE Computer*, 29(2):38–47, Feb. 1996.

[12] R. S. Sandhu, D. F. Ferraiolo, and D. R. Kuhn. The NIST Model for Role-Based Access Control: Towards a Unified Standard. In *Proceedings of the 5th ACM Workshop on Role-Based Access Control (RBAC'2000)*, Berlin, Germany, 2000.

[13] R. S. Sandhu and Q. Munawer. The ARBAC99 Model for Administration of Roles. In *Proceedings of the 15th Annual Computer Security Application Conference*, Scottsdale, AZ, Dec. 1999.

[14] C. M. Westphall and J. S. Fraga. Authorization Schemes for Large-Scale Systems Based on Java, CORBA and Web Security Models. In *Proceedings of the IEEE International Conference on Networks (ICON'99)*, pages 327–334, Brisbane, Australia, Sept. 1999.

[15] C. M. Westphall, J. S. Fraga, and M. S. Wangham. PoliCap—A Policy Service for CORBA Security Model. In *Proceedings of the 18th Brazilian Symposium on Computer Netorks*, pages 355–370, May 2000. (in Portuguese).