A Large-scale System Authorization Scheme Proposal integrating Java, CORBA and Web Security Models and a Discretionary Prototype

Carla Merkle Westphall and Joni da Silva Fraga
Federal University of Santa Catarina - LCMI-DAS-UFSC
Campus Universitário - Trindade - Florianópolis - SC - Brazil
PO Box 476 - CEP 88040-900
e-mail: {merkle, fraga}@lcmi.ufsc.br

Abstract - This paper presents an authorization scheme for large-scale networks that involves programming models and tools represented by Web, Java and CORBA. The authorization scheme is based on structures and concepts introduced in Web, Java and CORBA for security. A discretionary prototype is presented here, where the solutions adopted involving a concrete scheme are discussed. This scheme was developed in order to simplify authorization policy implementation in these systems. These policies are based on well-known literature security models as the Access Matrix Model and on the discretionary policy objects of CORBA Security Specification.

Keywords – Security, Authorization Policies, Authorization Schemes, CORBA.

1 INTRODUCTION

New services are appearing to support the increasing demand and the new necessities for large-scale distributed systems as the Internet. *Security* was always a very important question in these systems, even though it has been initially less considered in the evolution of large-scale networks and distributed applications in general. With the increase in the use of these networks in critical applications, such as the electronic commerce, security became a basic requirement in these systems.

On the other hand, this increasing demand of services and new applications in these systems have given rise to new paradigms and tools of distributed programming. According to this trend, a new generation of distributed applications also appears, whose distinct feature is the utilization of code mobility. These new approaches also introduce new problems of security as well as involving a need for new concepts and models of security suitable to this new context. Among these new paradigms and tools, besides code mobility, there is the Web and CORBA. Authorization schemes for these types of systems involving code mobility, therefore, must undergo a process of renewal and maturation, providing a broad subject for research in the security area.

The Java language popularized the concept of mobile code through its *applets* executed in Web browsers. The Web environment represents the simplest mobile code structure making it possible to load codes at any point of the network. The distributed programming model provided by Java, CORBA and Web is becoming a *de facto* standard of Internet programming [1]. The combination of automatic client code load and, also, the full independence of operational platform are great advantages for applications that use Java, CORBA and Web.

The purpose of this paper is to present the proposal of an authorization scheme for distributed applications in large-scale networks involving these new models of programming represented by the Web, Java and CORBA. The authorization scheme is based on structures and concepts introduced in Web, Java and CORBA, for security, and in well-known literature security models as the Access Matrix Model. Our idea is to propose a model that offers solutions for security problems found in the development of secure distributed applications in large-scale networks.

In this paper, section 2 presents Web, Java and CORBA security concepts. The authorization scheme considered for large-scale networks, combining Java, CORBA and Web security models, considering discretionary authorization policies is presented in section 3. The implementation results obtained with a first prototype are described in section 4. Related works are summarized in section 5. In the last part of the paper, conclusions on the results obtained are made and some future perspectives are indicated.

2 SECURITY MECHANISMS IN WEB, JAVA AND CORBA TOOLS

2.1 Web

The distributed programming model provided by the WWW environment is extremely powerful, scalable and useful in the development of distributed applications. The existence of a secure environment of this kind is necessary when the aim is to use it in different applications in a context of large-scale networks [2].

The evolution of the Web has added new features to this environment to satisfy the increasing demand, without considering carefully the impact in the security of the system. In general, the Web security problem can be divided into three parts: security of the server, security of the information in transit and security of the client.

The security of the server is based on authentication and access control services. The authentication services available are the *basic scheme*, the *scheme with digest authentication* and the *scheme with certificates* [2]. The basic scheme of authentication is an identification system based on providing a user name and a password. There is no concern in this system with providing any form of control that gives guarantees of the validity of the identification information provided. The digest authentication scheme provides some exchange mechanisms allowing authentication of the identification information by means of a digest calculated from this information. In the authentication in large-scale distributed systems, it is desirable that an intermediary trustworthy entity (certification authority) acts among the communicating pairs, by issuing and revoking certificates, in order to guarantee the information exchanges in the authentication from a client to a server. There are two ways of controlling access to the Web server: denying access to a client who desires a connection to the server, verifying her/his IP address, or, forbidding the access to a client until she/he produces some form of identification, typically a user name and the corresponding password. In this second approach, once connected, the user is subjected to the normal directory protection through access lists.

The security of the WWW also depends on the security of the information that passes through the Internet. The protection of this information in communication support involves what we could call cryptographic controls. The available cryptographic protocols for use in the Internet that can be mentioned are: SSL (Secure Socket Layer) and PGP (Pretty Good Privacy) [2].

Clients (browsers) can also be subjected to attacks. Programming languages and environments for Web, like Java and Javascript, with their new features imposed to improve Web interactivity, also raise new and additional security problems. Client security is treated in the context of Java language.

2.2 Java

In this article, mobile code relates to the software that travels through an heterogeneous network, crossing protection domains and being executed automatically at its destination. Security represents a great problem in the systems that provide support to code mobility and frequently is considered the main limitation for the broad use of these paradigms [3]. Java language popularized the notion of mobile code through its applets. The security model implemented for the Java platform, in its initial proposal, is centered in the *sandbox* concept [4]. The essence of the sandbox model is that the local code is trustworthy and has full access to the resources of the system (such as the file system) while the remote code (an applet) is not trustworthy and can have access only to limited resources, provided inside sandbox. This sandbox concept is used by the Java Development Kit (JDK) and generally is adopted by applications constructed under JDK, including Web browsers enabled to execute Java code.

The security in the Java platform is actualized on some levels. At first, an important part of this security model is in the actions of the compiler and the *bytecode verifier* that guarantee only the

execution of legitimate Java codes. The bytecode verifier inspects this code before its execution on the virtual machine (Java Virtual Machine - JVM), detecting the presence of potentially dangerous constructions, such as attributions of illegal data types. The bytecode verifier represents a static access control mechanism based on code inspection.

Also an important part of Java security is the *class loader*, a programmable tool that provides the means for retrieving and linking dynamically the classes of an application in a JVM, thus providing the idea of the language mobility.

On being executed, the access to the system resources is mediated by the JVM, through a *Security Manager* class, that restricts the actions of a non-trustworthy code to the minimum possible. This mechanism implements the access control (dynamic) in the code execution, materializing the sandbox concept.

Two phases of development can be cited from the original sandbox security model [4]. The first one involves kit JDK 1.1.x that introduced the *signed applet* concept. In this model, a digitally signed applet is treated as if it was a correct local code, since the signature's key is recognized as trustworthy for the system that received the applet for execution. Besides the signed applet concept, JDK 1.1.x defined a new API, called *Java Cryptography Architecture API*, projected to supply cryptographic functionalities to the applications developers in the Java platform [5]. In kit JDK 1.1, this cryptographic architecture includes classes and interfaces to provide digital signatures and message digests.

The second evolution of the Java security model is presented in kit JDK 1.2 where the following objectives are present: to provide fine-grained access control, to make security policies easily set, to define an access control structure that can be easily extended to all Java programs, including applications and applets. The concept of protection domains is fundamental in this new architecture, fulfilling in part these objectives mentioned. A domain can be defined as the set of objects that is directly accessed by a principal (in the security literature, a *principal* is defined as a user, a process or any active entity registered and authenticable in the system). Protection domains possess two distinct categories: system domains and application domains. It is important that all ways of access to the external resources, such as file systems, network services, screen and keyboard are accessible only through system domains. A system security policy, defined by the user or by the administrator of the system, must specify which protection domains must be created and which permission must be provided in these domains. The Java execution environment keeps a mapping of the code (classes and instances) for its protection domains and corresponding permission. In a way, the domain concept implements the least privilege principle. In JDK 1.2, an extension to the cryptographic API is also made, in order to support X.509v3 certificates [6].

2.3 CORBA

The CORBA/OMG specifications correspond to a set of standards and concepts for distributed objects in open environments considered by OMG (Object Management Group). These standards make it possible to have remote access to object methods, in a transparent form, in heterogeneous distributed environments through an ORB (Object Request Broker). The ORB, in a more generic meaning, is a communication channel for distributed objects.

The CORBA specifications define features of a support environment to distributed applications according to the OMA (Object Management Architecture) architecture. This architecture divides the distributed object space in three parts: application objects, constructed for the application programmer; service objects (COSS: Common Object Services Specification) that still support common services, regardless of application domains and, in Common Facilities that in turn, are services guided to application domains. All the objects in this architecture communicate via ORB.

The OMG wrote a document defining the main directions for distributed object security [7]. The security services of the CORBA form volume 3, being part of object services (COSS services) defined by the CORBA. The model described in this document establishes some procedures involving the authentication and the verification of the authorization, the invocation of a remote method, the security of the communication among objects, in addition to aspects involving schemes for delegating rights, non-repudiation, auditing and security management.

The CORBA security model relates objects and components on four levels of a system: application objects (application level), services objects, ORB services and the ORB core (all on the level of middleware CORBA), components of security technology (on the level of underlying security services) and components of basic protection, provided by a combination of the hardware and local operating systems. Figure 1 illustrates the levels and main components of CORBA security model, indicating the relationships among them. In this figure, the client and destination objects represent the applications level.

ORB services and service objects (COSS services) are constructed on the ORB core and extend the basic functions with additional qualities or controls, facilitating the distributed object implementation. A combination of ORB services and COSS services is used in the implementation of the CORBA security model. In the specifications of the CORBA security model, the ORB services are given the name of *interceptors*. Logically, an interceptor is interposed in the path of a call between a client and a destination. Each COSS service related to the security is associated with an interceptor, whose purpose is to cause the transparent deviation to the corresponding service.

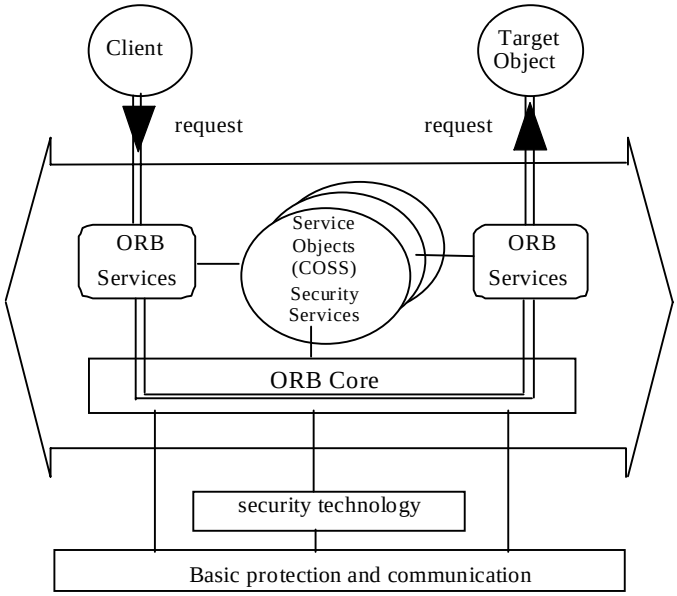

Fig. 1. CORBA security model.

In CORBA security model two interceptors are defined that act while a method is being requested: the *Access Control Interceptor* that on higher level causes a deviation to carry through the access control in the call and, the *Secure Invocation Interceptor* that makes a lower level interception in order to provide integrity and confidentiality properties in the corresponding invocation exchanges. These interceptors act, both in the client and in the server application object. The greatest box present in figure 2 shows the objects of the security model that act on an invocation as well as the respective *interceptors*. The *Access Control Interceptor* is represented by the object *Access Control* and the *Secure Invocation Interceptor* is represented by the *Secure Invocation* object.

The service objects that implement the security controls in CORBA specifications are: *PrincipalAuthenticator*, *Vault*, *Credential*, *DomainAccessPolicy*, *RequiredRights* and *AccessDecision* objects. The *Principal Authenticator* object corresponds to the CORBA principal authentication service. The *Vault* object establishes the secure associations between clients and servers with the respective security contexts. The *Credential* object represents the client credentials or rights in the session. The *DomainAccessPolicy* object represents the *discretionary* authorization policy management interface and grants a set of principals a specified set of rights to perform operations on all objects in the domain [8]. The *RequiredRights* object stores information about rights necessary to execute each method of each CORBA interface in the system. The *AccessDecision* object is responsible for interacting with *DomainAccessPolicy* and *RequiredRights* objects to determine if a given operation on a specific target object is permitted. A requestor will be granted authority to execute method *m* of object *o* if the credential associated with the request contains a privilege atribute that grants a right into a domain containing object *o* and if the same right is also included in the *RequiredRights* table of method *m*. There are other service objects related to non-repudiation and auditing.

The authentication described in the CORBA security specifications, defines a set of exchanges between the principal and the *Principal Authenticator* service object. These exchanges have as final aims the acquisition of credentials for the principal. An authenticated principal can obtain the necessary privileges so that it can have access to the system objects. These privileges are contained in a credential (*Credential* object). After acquiring credentials, the principal and the objects that act in its behalf, as clients, can begin to call server objects.

The interceptors that act in invocations of a method are created during the *binding* process between two application objects that are to communicate through these invocations. We distinguish the *initial invocation* when the binding process occurs between objects, of a *normal invocation* that represents any call among objects made after the establishment of the binding. A binding creates the context for secure communication between the communicating parts and is initiated when the client executes a binding operation. The *Access Control* object is created on the client and the server sides in order to execute the access control on both sides when invoking a method. The *Secure Invocation* object is responsible in binding time for the establishment of secure association between the client and the server. In binding time, this low-level interceptor activates the *Vault* service object in order to create a *Security Context* object of the secure association that must be established between client and server objects. The *Security Context* object stores cryptographic information of keys, algorithms, etc, used in this association.

The service objects in CORBA security model, in fact, isolate applications and the ORB of the security technology (figure 1), which consists of an underlying layer that implements some functionalities of related security service objects. Security technology includes authentication services, secure association services (distribution of keys, certificates, ciphers/deciphers), etc. Some technologies can be used to provide these services: SSL, SPKM (*Simple Public-Key GSS-API Mechanism*), Kerberos and CSI-ECMA [7]. This security technology can be accessed via generic security interfaces such as GSS-API (*Generic Security Services API*), that isolate the security service implementations of the functional details of the underlying services.

3 A LARGE-SCALE SYSTEM AUTHORIZATION SCHEME PROPOSAL INTEGRATING JAVA, CORBA AND WEB SECURITY MODELS

The integration of Java, CORBA and Web constitutes a powerful environment for the distributed programming in large-scale networks. This combination of tools forms a *Web of distributed objects* that allows it to explore the flexibility and the availability provided by the WWW technology: the user machine, regardless of its location on the world-wide network, needs only a browser to load the

client code. The user cannot be directly connected to a remote server, but simply interact or activate client software loaded in its computer browser. Client software can provide a graphic interface (GUI) that accepts orders from the user and makes the display of information. In this model the remote server (CORBA server) supplies a set of services that can vary from the simple access to the information up to the point of code execution. In the integration of these tools, the Java language contributes with mobile code support and CORBA with the integration technology of interoperable objects.

At first, in this environment, distributed applications are expressed in the form of clients represented by mobile codes or Java applets, and of servers implemented as distributed objects. When the user presents the URL of the desired service, he/she activates the load in the browser of the applet resident in the Web server machine. The applet code execution can contain calls to remote methods that are treated by the CORBA server in the server machine. The client (Java applet) interacts with a CORBA server (remote object) through an ORB, in a traditional client/server model[1].

The *authorization scheme* proposed in this item is to be used in the context of distributed applications in large-scale networks [9]. It is based on structures and concepts introduced in Web, Java and CORBA for security. In this authorization scheme two security control levels are defined: the *global level* and the *local level*. These two levels are actualized in CORBA service objects and in security nodes and TCB's (*Trusted Computing Bases*), respectively. The service objects concentrate functions of identification and authentication of users and authorization controls in the access of visible objects on the global level. The security nodes and TCB's, present in each machine of the system, validate the ways of access to the local resources. Figure 2 shows the main components that implement the controls of our scheme.

The authentication of users or principals in this scheme is done using an applet and a service object (figure 2). The authentication applet (mobile code) interacts with the *PrincipalAuthenticator* service object according to the exchanges specified by the CORBA for the identification and authentication of principals. When the user presents the URL of the desired service, she/he activates the load of this authentication applet. Once identification of the user is verified, establishing CORBA credentials for the client (service object named *Credential*), the authentication applet releases the corresponding URL, loading the application applet (CORBA client).

The application applet, initially, interacts with the CORBA name service (*CosNaming*) [10], to get, from the name of the object, the reference or IOR (*Interoperable Object Reference*) of the server application object. This must allow the binding with the server object. The calls executed by this application applet on a remote application server are subjected to two levels of access control. On the highest level, a *Policy Server* that holds two CORBA service objects named *DomainAccessPolicy* and *RequiredRights*, is responsible for the validation of access requests to the persistent objects, obtaining the rights in an access list, according to the appropriate policy. From this high level verification, capabilities are generated by the *ClientAccessDecision* object using the rights obtained from the *Policy Server*. This capabilities will be validated locally in the remote servers, completing, in this way, the second access control level defined in our scheme.

---

[1] Clients *Applets* are subjected to restrictions imposed by the browsers where they are executed. These restrictions prevent these codes from having access to the local disk or making connections with a different host than that from where they were loaded. This somehow imposes on the Web server machine, from where the client applet was loaded, the location of server application objects. Some techniques are offered to allow clients applets to free them from these restrictions and that allow them to comunicate with any objects in a distributed system without location limitation. We have adopted the *Wonderwall* solution [10], a proprietary solution of Iona, that forms a *software* located on the Web server site, acting as a message router from the applet to the application server. In this case, IIOP messages from the applet are transfered on the Web server machine to target objects on other sites.

To construct these two levels of access control, we use the two defined interception levels on CORBA security model, assigned in figure 2 as *Access Control* and *Secure Invocation* objects. The high level interception (*Access Control*), on the client side (application applet), deviates to the *Policy Server* object for the verifications of the access list. On the server side of the application, this high-level interception is used to validate the capability received with the invocation request. The validation of the capability is carried out by means of a local service object that is part of the TCB of the machine where the application server is located.

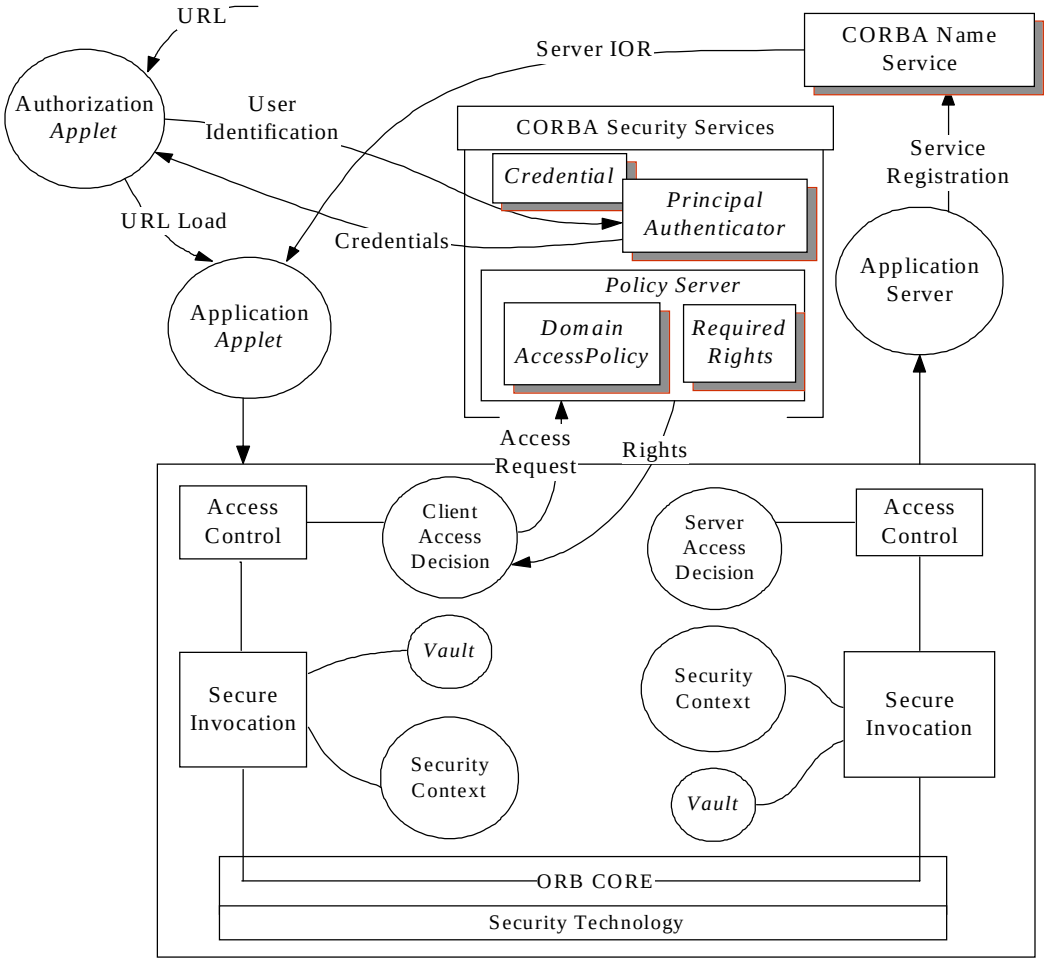

Fig. 2. Authorization Scheme Structure.

The low-level interception (*Secure Invocation*) of CORBA model, on both sides (client and server), is used to protect, in a secure association, the message that carries the request with its capability. The representation of privileges (or rights) in the form of capabilities in the authorization scheme uses CORBA structures, such as object references and credentials. Besides the cited controls above, the cryptographic controls are also necessary in the scheme and are defined in the CORBA service object form that uses the security technology resident under the ORB (*Vault* and *SecurityContext* objects).

The security nodes and *Trusted Computing Bases* validate the local accesses of the application applet. To implement this trustworthy base that validates the local accesses, we use the Java security model and its access control procedures. The Java security model, in its version 1.2, has file policies that identify which operations can be carried through by the loaded codes (applets). Moreover, it relies on the security manager that carries out the access control based on the policies actualized through the protection domains. To protect the host machine of a mobile code (applet), it is necessary for the mobile code to undergo an authentication phase and, during its execution, it is

necessary to validate the access to local resources. The authentication serves to guarantee the code origin. The privileges returned (credentials) from the user authentication determine the construction of the protection domain associated with the application applet activated by the user. These protection domains determine the dependence of these local forms of access according to the global policies.

The interactions of the *Access Control* object and the authentication applet with the *Policy Server* and *PrincipalAuthenticator* objects, respectively, take place through the ORB, using the secure associations defined. The URL submissions of a user can use the https protocol, establishing the use of the SSL and its cryptographic controls during the interactions with the Web server.

In large-scale networks the authorization process necessarily goes through the connection of a variety of name servers, each one responsible for a specific domain of objects and users. In each name domain, the controls of the authorization scheme must be present, centered in service objects pertinent to the considered domain. In other words, each name domain must have its *Policy Server* and *PrincipalAuthenticator* objects centering the global controls on persistent objects and users of the domain. The X.500 specifications provide means for these connections among different contexts of names based on *alias* mechanisms [11]. The *alias* can be understood as a local designation in a domain, identifying an object or user as non-local and it allows the search in resolving names to be extended to other domains. Initially, the authorization scheme covers a single domain, but to make feasible its use in large-scale networks, the possible use of LDAP (*Lightweight Directory Access Protocol*) - an implementation of directory services based on the X.500 and sufficiently used nowadays [12] - is envisioned.

4 IMPLEMENTATION RESULTS

A first prototype of the proposed scheme was developed in our laboratories. The implementations are made using the OrbixWeb 3.0 of the Iona [10], Netscape Communicator 4.5, JDK the 1.2 of the Sun and SSLeay - free version of SSL protocol 3.0 [13]. SSL (*Secure Socket Layer*) was chosen as a security technology, since it offers an available source code that can be readily modified, as a widely used protocol in Internet applications inserted by the OMG in its last security service revision [7] as one of the possible protocols to be used for applications that implement the CORBA security. In the prototype it was used the SSLeay 0.9.0b protocol.

This first experiment implements discretionary policies, being limited to a unique access control verification on the application server's side. This prototype is also limited to a name domain. Figure 3 synthecizes the functionalities implemented in the prototype.

Among objects implemented in this prototype, beyond the *Application Applet* and *Application Server* objects, we have the *Access Decision*, *Vault*, *Security Context*, *DomainAccessPolicy* and *RequiredRights* objects that are CORBA security model objects. These objects use other CORBA service objects (COSS), such as the name service.

The access control carried out in the prototype is based on an access list mechanism implemented by *DomainAccessPolicy* and *RequiredRights* objects. The entities that are subjected to the security controls in our system are the users (with their security privileges) and the objects. The access list summarizes the users and their execution rights over the object methods.

The access list is managed, in our prototype, by the *Policy Server* object that is responsible for the validation of access requests to the persistent objects, obtaining the rights in an access list, according to the appropriate policy. The access list is divided into two objects: the *RequiredRights* object and the *DomainAccessPolicy* object, both have their interface description presented in CORBA IDL in figure 4. The *RequiredRights* object stores the rights necessary for executing each method of each *object* in the system. The *RequiredRights* object has the following methods: *get_required_rights* and *set_required_rights*. The method *get_required_rights* retrieves the required rights to execute the specified method and the method *set_required_rights* updates the required

rights of the specified method of an interface. The *DomainAccessPolicy* object that stores the granted rights of each *subject* in the system (each subject is represented by an *privilege attribute*). The *DomainAccessPolicy* object is the *discretionary* management interface and has the following methods: *grant_rights*, *revoke_rights*, *replace_rights*, *get_rights* and *get_all_rights*. The method *grant_rights* provide the rights for the specified privilege attribute. For example, some bank manager object could grant rights of type 'modify some bank account' to user Bob that has a 'group' privilege. The method *revoke_rights* revokes rights, the method *replace_rights* changes old rights with the new rights specified, the method *get_rights* returns the rights granted to a privilege attribute and the method *get_all_rights* returns all rights granted to a privilege attribute. Insertion, updating and removal of rights methods are executed by the server objects owners, right after the servers have been registered in the name service.

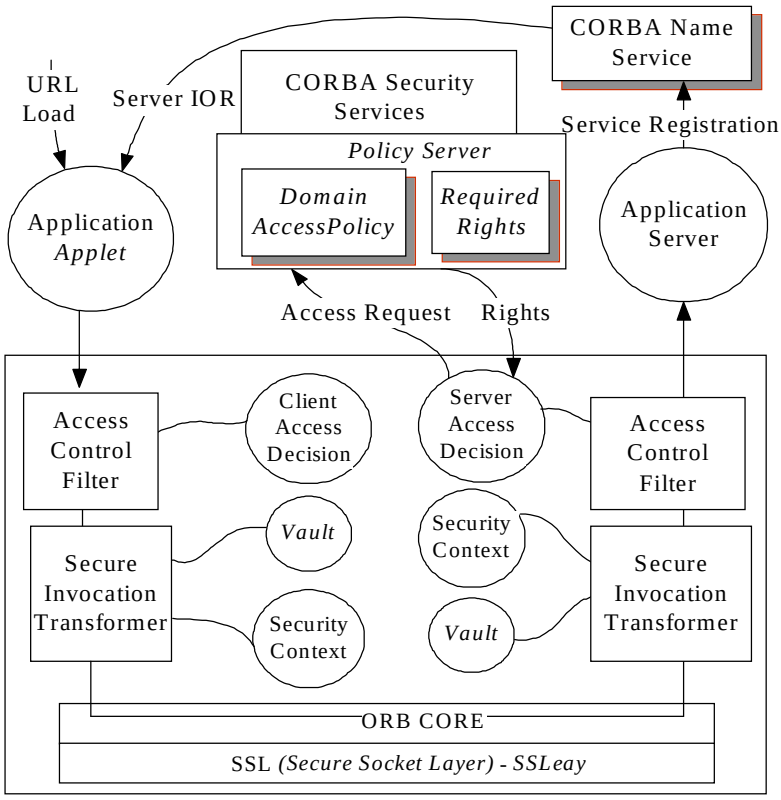

Fig. 3. Authorization Scheme Prototype Structure.

In order to simplify the interactions in the prototype, the *Policy Server* object was placed together with the CORBA application server and the Web server on the same site. In this way, the control using capabilities on the server site, present in the model of the previous section, becomes unnecessary since the global level control is carried out locally.

The implementations performed in the prototype use as deviation mechanisms the *filters*, present in the OrbixWeb, replacing the *interceptors* of the CORBA security model, which had still not been completely defined by the OMG. In the prototype, the only executing access control filter is placed on the application server site. This filter invokes the *ServerAccessDecision* object that, in turn, interacts with the *Policy Server* object for the access list verifications in the server machine.

Another type of interceptor present in the OrbixWeb is the *transformer*, that, unlike of the filter, can operate on request. In the prototype, transformers are used in the low-level interception. During the binding between client and server object, on the client site, the *Secure Invocation* transformer makes the *Vault* service object active in order to create the *SecurityContext* object,

establishing the secure association. The *Vault* object executes the SSL handshake between client and server and create the *SecurityContext* object that stores the information related to the established association.

During a normal invocation, on the client site, the Secure Invocation transformer activates the *SecurityContext* object in order to get information concerning keys, cryptographic algorithms, etc, used in the association. This information will be passed on to the transformer to execute the SSL cipher process on the IIOP message, before transmiting it via TCP. On the server side, the transformer object, having the *SecurityContext* information at hand, requests the SSL to carry out the deciphering of the message.

The *PrincipalAuthenticator* object (figure 2), responsible for the principal authentication and for the creation of the *Credential* object, was also not implemented in this initial prototype. The credentials in this prototype are created statically. Regarding applets authentication, the corresponding mechanisms had also not been implemented. These signature mechanisms and verification will be easily performed using the cryptographic API of JDK 1.2 [6].

```
// interface DomainAccessPolicy
interface DomainAccessPolicy : AccessPolicy {
  void grant_rights(
            in  Security::SecAttribute        priv_attr,
            in  Security::DelegationState     del_state,
            in  Security::ExtensibleFamily    rights_family,
            in  Security::RightsList          rights );
  void revoke_rights(
            in  Security::SecAttribute        priv_attr,
            in  Security::DelegationState     del_state,
            in  Security::ExtensibleFamily    rights_family,
            in  Security::RightsList          rights );
   void replace_rights (
            in  Security::SecAttribute        priv_attr,
            in  Security::DelegationState     del_state,
            in  Security::ExtensibleFamily    rights_family,
            in  Security::RightsList          rights );
   Security::RightsList get_rights (
            in  Security::SecAttribute        priv_attr,
            in  Security::DelegationState     del_state,
            in  Security::ExtensibleFamily    rights_family );
    Security::RightsList get_all_rights (
            in  Security::SecAttribute        priv_attr,
            in  Security::DelegationState     del_state );
};
```

```
// RequiredRights Interface
interface RequiredRights{
  void get_required_rights(
            in  Object                        obj,
            in  CORBA::Identifier             operation_name,
            in  CORBA::RepositoryId           interface_name,
            out Security::RightsList          rights,
            out Security::RightsCombinator    rights_combinator );
  void set_required_rights(
            in  string                        operation_name,
            in  CORBA::RepositoryId           interface_name,
            in  Security::RightsList          rights,
            in  Security::RightsCombinator    rights_combinator );
};
```

Fig. 4. IDL specification of the CORBA *DomainAccessPolicy* and *RequiredRights* objects.

The mechanism of capabilities in our scheme is being specified. The actualization of this mechanism will have to make use of the *Request* sent by the client when an invocation of a remote method is made. As a *Request* is composed of the server object IOR and of a field indicating the requested method, the simple addition of the user rights forms a *ticket* characterizing a *capability*, that in its simpler concept contains the name of the target object and the rights of the owner of this ticket over the object. The access control filter provide ways of modification of a request (*Request* object). In this way, in the flow of a call execution, after the *ClientAccessDecision* object obtained from the *Policy Server* object the referring rights of the call, these rights will be inserted into the *Request* by the access control filter. The capability, as well as all the important fields of the *Request*, will be ciphered in the transformer before being sent. On the server side, the transformer deciphers the data, recovering the capability so that it can be validated by the *ServerAccessDecision* object. To prevent replay attacks it is necessary to have mechanisms to include *nonce* fields in the capability. These nonces may have been negotiated, during the establishment of the secure association to become available from the *SecurityContext* object. The presence of a capability mechanism will make our scheme take on again its original form, presented in figure 2, where the

Policy Server object is located in a different machine of the application server and is activated from the client site.

5 RELATED WORK

In the literature, in general, there are a small number of experiences exploring security models and concepts introduced in CORBA and Java specifications. Even rarer are studies involving the integration of the concepts of these tools. The proper OMG makes an only reference in [14], to the combination of the code mobility concept with the CORBA security model. The idea in this document is to extend the supports of mobile agents with the available CORBA security service controls.

An academic experience involving security and CORBA can be found in [15]. In this work a proposal is made of a security model for distributed objects supported for the CORBA. The main security services dealt with in the paper are client and server authentication, access control and protection services. In this model, access control is performed in the form of a distributed reference monitor that examines each client access request with control information from the server object. This reference monitor constitutes a layer between the ORB core and its interfaces, and is called *Object Security Controller* (OSC). In this model the notion of *ORB nodes*, machines shared for objects and clients, is introduced. Each ORB node stores control information of its objects in the form of access lists. Control information from a name service is available in a global form. The paper discusses its proposal assuming authentication mechanisms based on public-key. Likewise, the creation of forms of dynamic object and security information are presented. This experience is not based on CORBA security specifications. The distribution of the security information (access list, keys, etc) is different from our approach and it seems to be much more vulnerable.

Some products implementing CORBA security model can be mentioned [16] such as: the OrbixSecurity of the Iona Technologies company [17], DAIS of the PeerLogic company and ORBAsec SL2 of the Adiron company. With the exception of the OrbixSecurity that implements CORBA security model level 1, all the others, including our own work, follow level 2 that provides a more complete set of functionalities. These products adopt GSS-API standards and the Kerberos as underlying security technology. The access control in these products is only made in the application server. This makes it difficult to use these products to reach the implementation goal of our global policies. There are other available products and software that combine CORBA with SSL. The OrbixSSL of the Iona, the Visibroker SSL of the Inprise, the ORBacus SSL of OOC company and the SSL support of MICO ORB [18] are examples of this combination.

6 FINAL CONSIDERATIONS

The combination of the automatic client code load and the full operational platform independence, make the integration of the Java, CORBA and Web tools highly desirable to the distributed object programming in the Internet. The objective of this paper was to present the proposal of a security scheme for distributed applications in large-scale networks. This authorization scheme was conceived in order to be feasible, practical and with policies that can be followed and easily defined in large-scale distributed applications. The scheme presented is based on structures and concepts introduced for security by the above mentioned tools.

In this authorization scheme two security controls are defined: a global level and a local level. These two levels are performed by CORBA service objects and by security nodes and local TCB's. The service objects combine identification and user authentication functions and cryptographic authorization controls for access visible objects on the global level. The security nodes and TCB's, present in each system machine, validate the applets access to the local resources.

The CORBA service objects are conceived in order to be located in non-shared and secure machines in the network. One possibility is the location on the Web server site, considering this a

secure host of the network or of the name domain considered. The centralization of control information in these service objects, defining the global level of the authorization scheme, minimizes the impact on the violation of a security node or of another machine of the system considered.

A prototype is being constructed in order to verify the viability of the considered authorization scheme. In its current version, this prototype is quite simplified. In this version, the service objects, the application objects and the Web server share the same machine. Here we implement only discretionary policies, making a single access control verification on the application server side. This prototype, with the tests performed, was useful to give us guarantees concerning certain implementation options of the scheme. Basically, we limit ourselves, in these implementations, to performing the control mechanisms of the interactions between application applets and application server objects. The security mechanisms that act during these interactions had been conceived to be totally transparent to the application.

One of the objectives of the authorization scheme is to allow the implementation of security policies in a simple way. Discretionary policies, to a certain extent, had already been implemented in the experiments performed up to now. Non-discretionary or mandatory policies are the future goals of our prototype. Using the *Credential* and *CORBA access control objects* of the scheme we believe that we can still implement a version of the Bell and Lapadula model [19] or of the Role-based Access Control models [20] [21].

REFERENCES

[1] Normalisation – Java en route vers le standard ISO, Le Monde Informatique, n. 743, 21 novembre 1997.

[2] A. D. Rubin, D. Geer and M. Ranum, "Web Security Sourcebook," *John Wiley & Sons*, 1997.

[3] T. Thorn, "Programming Languages for Mobile Code," *ACM Computing Surveys*, vol. 29(3):213-239, Sep. 1997.

[4] S. M. Inc., "Java Security Architecture (JDK 1.2) Document Version 1.0," *Sun M. Inc.*, Oct. 1998.

[5] S. M. Inc., "Java Cryptography Architecture API Specification & Reference," Sun M. Inc., May 1997.

[6] S. M. Inc., "Java Cryptography Architecture API Specification & Reference," Sun M. Inc., Oct. 1998.

[7] OMG, "Security Service:v1.2 Final," *OMG Document Number 98-01-02,* Nov. 1998.

[8] G. Karjoth, "Authorization in CORBA Security," In *Proceedings of the Fifth ESORICS*, Lecture Notes in Computer Science, pp. 143-158, Springer-Verlag, Berlin Germany, September 1998.

[9] C. M. Westphall, "Authorization Schemes for Distributed Programming based on Java/CORBA/Web Security Models," *Doctor Exam*, LCMI-UFSC, Certificate no. 165.863, book 277, page 4, Rio de Janeiro, Brazil, Aug. 1998.

[10] IONA Technologies, "OrbixWeb Programmer's Guide," *IONA Technologies*, 1997.

[11] ITU-T. "Autentication Framework," *ITU-T Recommendation X.509*, Nov. 1993.

[12] D. Kosiur, "LDAP: The next-generation directory?,", *SunWorld Online*, Oct. 1996. http://www.sunworld.com/

[13] A. O. Freier, P. Karlton and P. C. Kocher, "Secure Socket Layer 3.0," *Internet Draft*, Nov. 1996.

[14] OMG, "Mobile Agent System Interoperability Facilities Specification," *orbos/97-10-05*, Nov. 1997.

[15] R. H. Deng, S. K. Bhonsle, W. Wang and A. Lazar, "Integrating Security in CORBA Based Object Architectures," *Proc. of IEEE Symposium on Research in Security and Privacy*, pp. 50-61, Oakland, CA, May 1995.

[16] U. Lang, "Current State of CORBA Security in Practice – CORBA Security Service Implementations and other CORBA Security Products," *University of Cambridge – Computer Laboratory*, 1998.

[17] IONA Technologies, "OrbixSecurity v1.0 White Paper," *IONA Technologies*, 1997.

[18] "What is Mico ?," Feb. 1997. http://diamant-enl.vsb.cs.uni-frankfurt.de/~mico/

[19] D. E. Bell and L. J. Lapadula, "Secure computer systems: Mathematical foundations and model," Tech. Rep. M74-244, MITRE Corp, October 1974.

[20] R. S. Sandhu, "Role-Based Access Control," *Advances in Computer Science*, vol. 46, Academic Press, 1998.

[21] K. Beznosov and Y. Deng, "A Framework for Implementing Role-based Access Control Using CORBA Security Service," In *Proceedings of 4rd ACM Workshop on Role-Based Access*, October, 1999.