

JaCoWeb Security - A CORBA Security Discretionary Prototype

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

Abstract

This paper presents a CORBA Security discretionary prototype developed in the context of JaCoWeb Security Project. JaCoWeb Security Project is developing an authorization scheme for large-scale networks that is based on structures and concepts introduced in Web, Java and CORBA for security. This scheme is being developed in order to deal with management of security policies in large-scale networks simplifying authorization policy implementation. 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

Management of security policies in large-scale systems, such as Internet, is a big concern today. However, it presents several difficulties, as there is a broad number of users, objects and operations and a lack of security policies enforcement and heterogeneous environments present in large-scale networks. Therefore, complexity and scaling difficult security policies management.

As businesses today are increasingly dependent on their information systems, there is a growing demand for information security. However, in distributed or large-scale systems, it is not easy to achieve protection of business information because physical protection of the computers can hardly ever be provided. Also, since information has to be shared, technically everyone can get access to systems through a network. The inherent complexities of distributed object systems cause additional vulnerabilities, which have to be prevented by the security architecture. Among these vulnerabilities, the access control on very large systems is problematic, since distributed object systems can scale without limit and new components are constantly added, deleted and modified in these environments. In geographically large systems there are usually many different security policy domains that difficult their administration. To solve these problems, there is the CORBA Security Service that, if implemented and administered properly, can provide a high level of security for information and applications in large-scale environments.

The increasing demand for services and new applications in large-scale systems have given rise to new distributed programming paradigms and tools. According to this trend, a new generation of distributed

applications also appears, whose distinct feature is the utilization of code mobility. Among these new paradigms and tools, besides code mobility, there is the Web and CORBA. Authorization schemes for these types of systems, therefore, must undergo a process of renewal and maturation, providing a broad subject for research in the security area.

The Java language made popular 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 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 *JaCoWeb Security Project* (<http://www.lcmi.ufsc.br/jacoweb/>), in development in our laboratories, aims to understand and integrate CORBA security model with Web and Java security models to compose an authorization scheme for distributed applications in large-scale networks. The authorization scheme is based on structures and concepts, for security, introduced in Web, Java and CORBA and in well-known security models, such as the Access Matrix Model. We propose a model that offers solutions for security problems present in the development of secure distributed applications in large-scale networks.

The aim of this paper is to present the *JaCoWeb* Authorization Scheme and the implemented CORBA Security discretionary prototype. In section 2, concepts of security policies and authorization schemes are presented. Section 3 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 the prototype are described in section 4. Related works are summarized in section 5. In the last section of the paper, conclusions on the results obtained are made and some future perspectives are indicated.

2 Security Models

The concept of security in a computer system is identified with its capacity to assure the prevention of the illegitimate access and the manipulation of the information or, further, to prevent the improper interference in its normal operation [2]. This capacity is based on four properties that must be kept: confidentiality, availability, integrity and authenticity. To assure the security of a system is then a huge task, considering the dimensions of the current systems.

The set of security properties that should be assured in a system and the way these properties are guaranteed are strongly linked to the definition of the system's security policies. The *security policy* or the *authorization policy* of a system is the set of rules and rights that determine the way the information and the other resources are managed, protected and distributed in a specific system. The authorization policies are generally described making use of security models [3]. Important literature security models that can be mentioned are the Access Matrix Model, Bell and Lapadula and Role-based Access Control Models. The policies are classified in two categories: discretionary and mandatory or non-discretionary policies.

In the case of the discretionary policies, the responsible for the resource (generally the resource owner), manipulates the access rights of each resource without restraint, according to his/her purpose (to his/her discretion). The Access Matrix Model, introduced by Lampson [3], describes discretionary authorization policies and is based on the concepts of subjects, objects and rights. A subject has the access right over an object being able to execute the corresponding operation on the object, if this right is expressed in the access matrix.

The mandatory or non-discretionary policies summarize in its authorization schemes, a set of unavoidable rules that express a type of organization involving the information security in the system as a whole. The mandatory policies assume that the users and objects or resources of the system, are all labeled; the labels of objects follow a specific classification while the users or the subjects of the access have clearance levels. The controls that determine the access authorizations are based on a matching of the clearance of the user with the classification of the object. The rules defined in these controls and considered to be unavoidable, make sure that the system verifies the integrity and confidentiality properties. The lattice-based models describe non-discretionary authorization policies. One of the important examples of lattice-based models is the Bell and Lapadula model of the DoD (Department of Defense of the United States) [3].

The mandatory authorization policies are used jointly with the discretionary ones. In this case, a user is authorized to manipulate an item of information if she/he has the right to the corresponding access (discretionary control) and if he/she is qualified for the level of the information classification (mandatory control). The discretionary and mandatory policies are recognized in official standards of secure systems classification as the *Common Criteria (CC)*. CC is the new ISO standard of security evaluation that was developed to fulfill necessities of all countries that deal with security evaluation [4].

2.1 Authorization schemes in distributed systems

An *authorization scheme* is the actualization of the authorization policies through a set of mechanisms. These mechanisms assure that all forms of access to the objects in the system are authorized by the defined policies. As for access control, these mechanisms are implemented by a set of hardware and software resources forming what we could call *security node*. The implementation of an authorization project does not depend only on access control. Other internal controls are also important, such as the cryptographic controls, authentication services, identification services etc. These additional controls form what the TCSEC (*Trusted Computer System Evaluation Criteria*) identifies as TCB (*Trusted Computing Base*). The security node and the TCB are two ideas introduced by the TCSEC as essential for the construction of secure systems.

Considering distributed systems, it can be stated that there are centralized approaches in the implementation of authorization schemes, and approaches based on the distribution of security functions in the system. Among the approaches based on the distribution of system security functions, there are approaches like Kerberos [5], Delta-4 [6] and X.509 [7]. In Kerberos approach, the authentication of the system is managed by a single trustworthy server (the Kerberos), while each site independently controls the authorization in the system. This approach presents difficulties in keeping the coherence of the authorization policies since the access matrix is distributed over all network sites (some servers deciding concurrently on the ways of accessing the system). The single authentication server is a vulnerable point of the system.

In the Delta-4 approach a quorum of authentication sites is defined, called in the original security literature as *security sites* [6]. In this approach, a client has access to the objects in servers after obtaining the authentication in a majority of the security servers. The security sites that form the authentication *quorum* are also responsible for the management of the persistent object accesses in the system.

Considering large-scale networks, like the Internet, an access control of the persistent objects is difficult on a global level. The global level functions in these systems are normally limited to an authentication server in a hierarchic form (X.509). In the X.509 approach, ITU-T standard [7], the authentication service is partitioned in order to increase its applicability in large-scale networks as the Internet. Different certification authorities (CA - Certification Authority) are arranged in the form of a tree. A particular client is registered in one of these CA's. The client access to objects of a server registered in another CA involves the interaction between the client and server CA's during the authentication process.

3 Security Mechanisms in Web, Java and Corba Tools

3.1 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 WWW secure environment is important when the aim is to use it in different applications in a large-scale network context [8].

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* [8]. 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, denying 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. One of the available cryptographic protocols used in the Internet that can be mentioned is the SSL (*Secure Socket Layer*) [8].

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.

3.2 Java

In this article, mobile code relates to the software that travels through a 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 [9]. Java language popularized the notion of mobile code through its applets. The security model implemented for the Java platform, in its initial proposal, is centralized in the *sandbox* concept [10]. 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 [10]. The first one involves JDK 1.1.x kit 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*, designed to supply cryptographic functions to the application developers in the Java platform [11]. In JDK 1.1 kit, 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 JDK 1.2 kit 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 the mentioned objectives. 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 authentic in the

system). Protection domains possess two distinct categories: system domains and application domains. It is important that all ways of accessing external resources, such as file systems, network services, screen and keyboard can only do so 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 permissions 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 [12].

3.3 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 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 [13]. The security service of the CORBA is part of object services (COSS services). The security model 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 application 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, *interceptors* implement the ORB services. Logically, an interceptor is interposed in the path of a call between a client and a destination. Each COSS service related to security is associated with an interceptor, whose purpose is to cause the transparent deviation to the corresponding service.

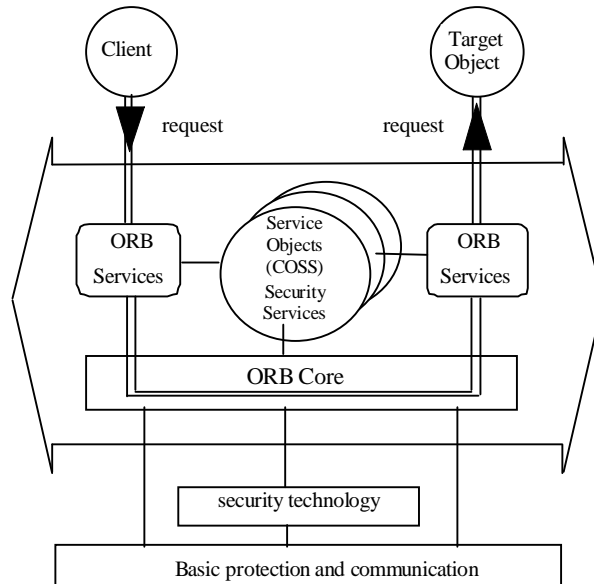


Fig. 1. CORBA security model.

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 establish a secure association and provide integrity and confidentiality properties in the corresponding invocation exchanges. These interceptors act, both in the client and in the server application object. The largest 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 [14]. 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 attribute 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 on its behalf, as clients, can begin to call server objects.

The interceptors that act on 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 a 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 *SecurityContext* object of the secure association that must be established between client and server objects. The *SecurityContext* object provides the security context information and is used by the *Secure Invocation* object in protecting messages for integrity and/or confidentiality.

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 implement some functions of the 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 [13]. This security technology can be accessed via generic security interfaces such as GSS-API (*Generic Security Services API*), which isolate the security service implementations of the functional details of the underlying services.

4 JaCoWeb Security - A Large-Scale System Authorization Scheme

The *JaCoWeb* Security Project (<http://www.lcmi.ufsc.br/jacoweb/>), in development in our laboratories, aims to understand and integrate CORBA security model with Web and Java security models to compose an authorization scheme for distributed applications in large-scale networks. Our idea is to propose a model that offers solutions for security problems present in the development of secure distributed applications in large-scale networks.

The integration of Java, CORBA and Web constitutes a powerful environment for the distributed programming in large-scale networks. It allows the exploration of the flexibility and the availability provided by the WWW technology: the user machine, regardless of its location on the worldwide network, needs only a browser to load the client code. The user could not 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 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. The CORBA platform can be used for integrating systems and applications, supplying the flexibility required by Web dynamically changing business environments.

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 (figure 2). 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.

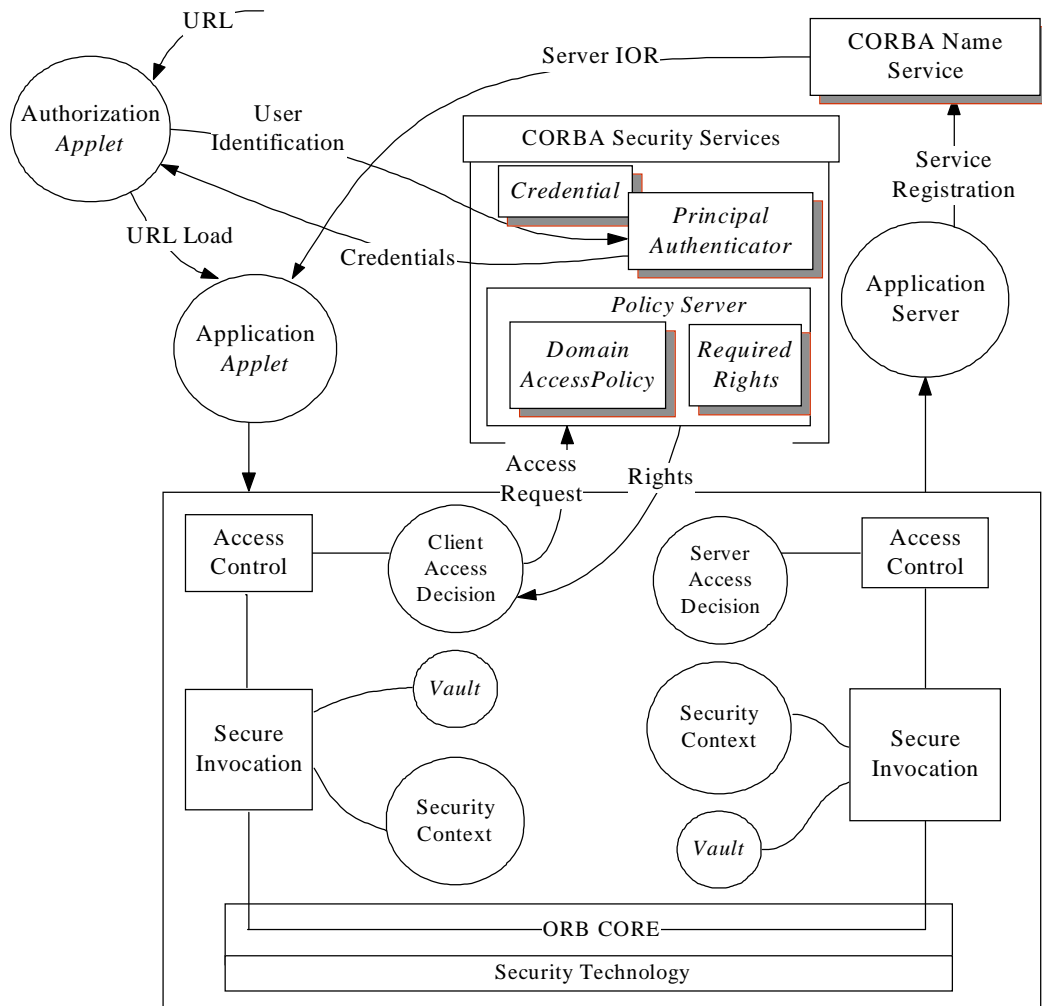


Fig. 2. Authorization Scheme Structure.

The *authorization scheme* proposed in this item is to be used in the context of distributed applications in large-scale networks [15]. 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, 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 accessing 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*) [16], 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, the *ClientAccessDecision* object using the rights obtained from the *Policy Server* generates capabilities. These capabilities will be validated locally in the remote servers, completing, in this way, the second access control level defined in our scheme.

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.

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 defined secure associations. 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 connections 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, centralized 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 [17]. The alias can be understood as a local designation in a domain, identifying an object or user as non-local, and it allows the resolving names search 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 [18] - is envisioned.

5 Implementation Results

A prototype of the proposed scheme was developed in our laboratories. The example used is a banking-system prototype composed by a CORBA server object and a Java client applet. The CORBA server object was developed with the JacORB 1.0¹ tool [16], a free Java ORB, and the Java client applet with the JDK 1.2.1 programming tool. The Netscape Communicator 4.5 web browser is also used as the environment from within which the client and server interact.

The purposes of this implementation are to carry out a discretionary access control based on an access control list, and to assure security of transmitted messages using as underlying security technology the SSL protocol - *Secure Socket Layer* [20]. These objectives are reached using CORBA security objects, the Java security model and Netscape Communicator 4.5 web browsers.

SSL was chosen as the security technology, mainly because it offers an available code in Java that can be readily integrated with a Java ORB, it is a widely used protocol in Internet applications, and was inserted by the OMG in its last security service revision [13] as one of the possible protocols to be used for applications that implement the CORBA security. In the prototype we used the IAIK-JCE for cryptographic

¹ In our first prototype [21] OrbixWeb 3.0 of Iona was used. Because it is a proprietary solution, it disallows extensions, limiting our implementation. The JacORB [16] comes with full source code.

functions and the iSaSiLk v.5.2 package² that implements SSL v3 in Java (available from University of Graz in Austria [19]).

Moreover, digital signature was used to allow clients to have access to the local disk or to make connections with a host different than that from where they were loaded. This technique frees applets from restrictions imposed by the Java sandbox model and allows them to communicate with any objects in a distributed system without location limitations.

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

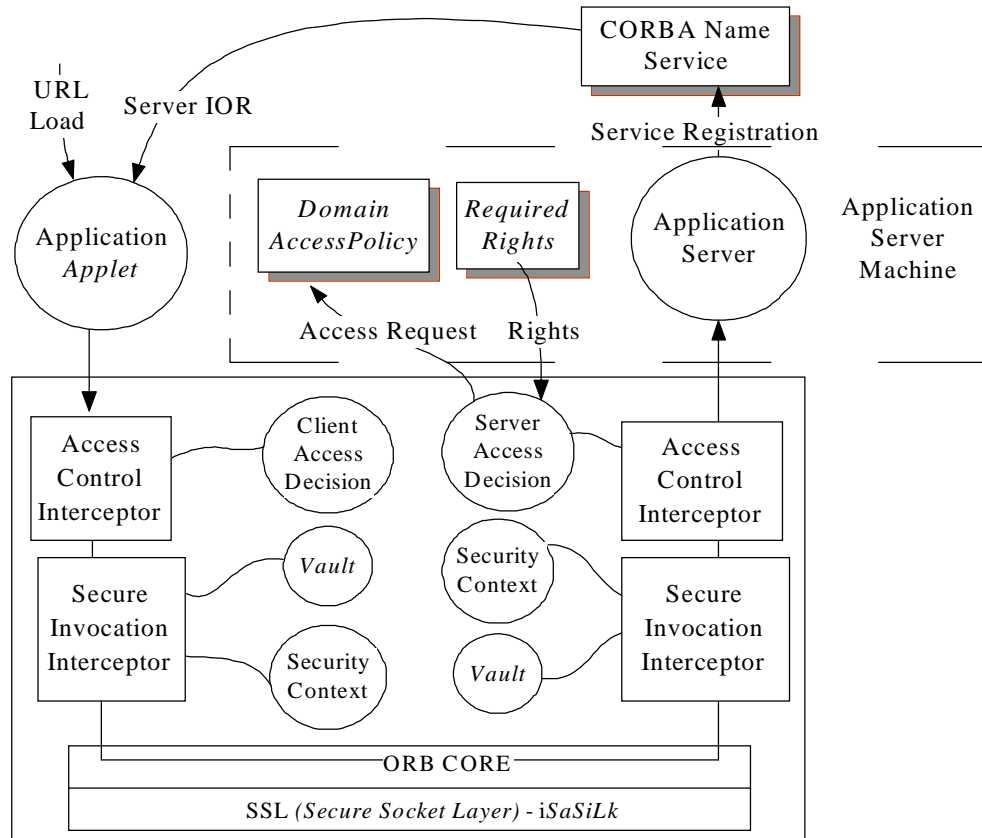


Fig. 3. Authorization Scheme Prototype Structure.

² The SSLeay, adopted in the first prototype [21], is written in C. It was replaced by iSaSiLk [19] that is implemented in Java and was easily adapted and inserted in the JacORB (also a Java implementation).

Among the objects implemented in this prototype (besides the *Application Applet* and *Application Server* objects) are the *AccessDecision*, *Vault*, *SecurityContext*, *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 the *DomainAccessPolicy* and the *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 *AccessDecision* 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.

<pre>// 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); };</pre>	<pre>// 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); };</pre>
--	---

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

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 stores the granted rights of each subject in the system (each subject is represented by a 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* provides 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, who has a

'group' privilege. The method *revoke_rights* revokes rights; the method *replace_rights* changes old rights with the new rights specified and the method *get_rights* returns the rights granted to a privilege attribute. The method *get_all_rights* returns all rights granted to a privilege attribute. The server objects owners execute insertion, updating and removal of rights, right after the servers have been registered in the name service.

In order to simplify the interactions in the prototype, the *DomainAccessPolicy* and *RequiredRights* objects were placed together with the CORBA application on the same site. Therefore, the access control using capabilities on the server site, as shown 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 *interceptors*, present in the JacORB. In the prototype, the only executing access control interceptor is placed on the application server site. The *Access Control Interceptor* invokes the *ServerAccessDecision* object that, in turn, interacts with the *RequiredRights* and *DomainAccessPolicy* objects for the access list verifications in the server machine.

According to the CORBA Security model [13], during the binding between client and server objects, the *Secure Invocation Interceptor* establishes the secure association. During a normal invocation, the Secure Invocation Interceptor activates the *SecurityContext* object to protect and reclaim the messages.

The entire negotiation and use of SSL, in the prototype, is performed transparently by iSaSiLk. This package's API (Application Programming Interface) disallows the *Vault* object from having control over the SSL *handshake*, and also the *SecurityContext* object from protecting messages using the established SSL context. Since CORBAsec specifications do not present a standard way (regarding concepts and needs) to integrate 'pluggable' security technologies, such as SSL, and the ORB (*Object Request Broker*), this work fulfill the CORBA security model [13].

The *PrincipalAuthenticator* object (figure 2), responsible for the principal authentication and for the creation of the *Credential* object, was not implemented in this initial prototype either. The credentials in this prototype are created statically having security attributes that will identify what type of rights a client has in the system. Security attributes are represented by privilege attributes, as an access identifier (*AccessId*) or a role identifier of the client in the system (*Role* can take on values such as *ManagerOfBank* or *ClientOfBank*, for example), and by the underlying security identifier (SSL).

As an evolution of our first prototype [21], by using signed applets, Web server and application server do not necessarily have to be kept in the same machine. Netscape Communicator 4.5 uses Java Plug-in 1.2 providing the ability to sign applets using Java Security API. The Java system security policy, described in a policy file, was defined in order to grant permissions to the client applet, allowing this applet to communicate with the name service and application server (respecting the structure of interactions represented in figure 3).

The capabilities mechanism in our scheme is currently 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. Since 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 simplest concept contains the name of the target object and the rights of the owner of this ticket over the object. The access control interceptor provides ways of modification of a request (*Request* object). So, in the flow of a call

execution, the *ClientAccessDecision* object will obtain from the *Policy Server* object the referring rights of the call, and the access control interceptor will insert these rights into the Request. The capability, as well as all the important fields of the *Request*, will be ciphered in the secure invocation interceptor before being sent. On the server side, the secure invocation interceptor deciphers the data, recovering the capability so that the *ServerAccessDecision* object can validate it. 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 its original form again, as 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.

The development of the prototype is a real CORBA security discretionary implementation (there are few experiments of this kind available today). It is important to note that CORBA security model defines a wide scope of abstractions, however they are not well described and the understanding of the dynamic model in a real application demand a considerable effort. So, we consider that the prototype development dealt with important questions, such as the definition of dynamic aspects of the CORBA security model, present in the construction of a real application. Besides, security policies, in a distributed application developed to be used in a large-scale context, are enforced transparently to client and server application objects.

The complexity and scaling problems, present in security policy management, were treated using the integration of the Java, CORBA and Web security models. The prototype facilitates security management and administration of security policies allowing security enforcement on ORB level. ORB-enforced security has advantages such as ORBs can be trusted to enforce security while applications cannot, and security enforcement for security-unaware objects is possible.

Security purposes such as integrity, confidentiality and mutual authentication were solved using SSL protocol as underlying security technology. These security objectives are important features in the application considered (a banking-system).

6 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 a single reference in [22], to the combination of the code mobility concept with the CORBA security model. The idea in this document is to extend the support for mobile agents with the available CORBA security service controls.

An academic experience involving security and CORBA can be found in [23]. 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, it presents the creation of forms of dynamic object and security information. This experience is not based on CORBA security specifications.

Some products implementing CORBA security model can be mentioned [24] such as: the OrbixSecurity of the Iona Technologies company [25], DAIS of the PeerLogic company and ORBAssec 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, Kerberos and SSL 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 [26] are examples of this combination.

7 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 accessing 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. 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 was constructed in order to verify the viability of the considered authorization scheme. In this version, the service objects, the application objects and the Web server do not necessarily have to share the same machine. Here we implement only discretionary policies, making 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 [27] [14] or of the Role-based Access Control models [28] [29].

REFERENCES

- [1] Normalisation – Java en route vers le standard ISO, *Le Monde Informatique*, n. 743, 21 novembre 1997.
- [2] V. Nicomette, “La Protection dans les Systèmes à Objets Répartis,”. *PhD thesis*, Institut National Polytechnique de Toulouse, 1996.
- [3] C. E. Landwehr, “Formal models for computer security,” *ACM Computing Surveys*, vol. 13, no. 3, pp. 247-278, Sep. 1981.
- [4] ISO/IEC. 15408-1:1999(E), "Part 1: Introduction and general model," *In: Information Technology – Security Techniques - Evaluation Criteria for IT Security*, First edition, ISO/IEC, December 1999.
- [5] B. Clifford Neuman and Theodore Ts'o, "Kerberos: An Authentication Service for Computer Networks," *IEEE Communications*, vol. 32(9), September 1994.
- [6] J. S. Fraga, “La Sécurité des Données par la Tolérance aux Intrusions,” *PhD thesis*, Institut National Polytechnique de Toulouse, 1985.
- [7] ITU-T. “Information Technology - The Open Systems Interconnection - The Directory: Authentication Framework,” *ITU-T Recommendation X.509*, Nov. 1993.
- [8] A. D. Rubin, D. Geer and M. Ranum, “Web Security Sourcebook,” *John Wiley & Sons*, 1997.
- [9] T. Thorn, “Programming Languages for Mobile Code,” *ACM Computing Surveys*, vol. 29(3):213-239, Sep. 1997.
- [10] S. M. Inc., “Java Security Architecture (JDK 1.2) Document Version 1.0,” *Sun M. Inc.*, Oct. 1998.
- [11] S. M. Inc., “Java Cryptography Architecture API Specification & Reference,” *Sun M. Inc.*, May 1997.
- [12] S. M. Inc., “Java Cryptography Architecture API Specification & Reference,” *Sun M. Inc.*, Oct. 1998.
- [13] OMG, “Security Service:v1.2 Final,” *OMG Document Number 98-01-02*, Nov. 1998.
- [14] 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.
- [15] C. M. Westphall, “An Authorization Scheme for Security in Large-Scale Distributed Systems,” *PhD Thesis*, UFSC-PGEEL-LCMI, Florianópolis, Santa Catarina, Brazil, Dec. 2000.
- [16] Gerald Brose, “JacORB – A free Java ORB,” *Freie Universität Berlin*, Institut für Informatik, Berlin, 1999 (<http://www.inf.fu-berlin.de/~brose/jacorb/>).
- [17] ITU-T. “Authentication Framework,” *ITU-T Recommendation X.509*, Nov. 1993.
- [18] D. Kosiur, “LDAP: The next-generation directory?,” *SunWorld Online*, Oct. 1996. <http://www.sunworld.com/>
- [19] Graz - University of Technology, “iSaSiLk Toolkit,” *Institute for Applied Information Processing and Communications*, Graz - University of Technology, 1999 (<http://jcewww.iaik.at/iSaSiLk/isasilk.htm>).
- [20] A. O. Freier, P. Karlton and P. C. Kocher, “Secure Socket Layer 3.0,” *Internet Draft*, Nov. 1996.
- [21] Carla Merkle Westphall and Joni da Silva Fraga, “Authorization Schemes for Large-Scale Systems based on Java, CORBA and Web Security Models,” *ICON 99 - The IEEE International Conference on Networks*, September 28 to October 1, pp. 327-334, 1999, Brisbane-Queensland, Australia.
- [22] OMG, “Mobile Agent System Interoperability Facilities Specification,” *orbos/97-10-05*, Nov. 1997.

- [23] 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.
- [24] U. Lang, "Current State of CORBA Security in Practice – CORBA Security Service Implementations and other CORBA Security Products," *University of Cambridge – Computer Laboratory*, 1998.
- [25] IONA Technologies, "OrbixSecurity v1.0 White Paper," *IONA Technologies*, 1997.
- [26] "What is Mico ?," Feb. 1997. <http://diamant-enl.vsb.cs.uni-frankfurt.de/~mico/>
- [27] D. E. Bell and L. J. Lapadula, "Secure computer systems: Mathematical foundations and model," *Tech. Rep. M74-244*, MITRE Corp, October 1974.
- [28] R. S. Sandhu, "Role-Based Access Control," *Advances in CS*, vol. 46, Academic Press, 1998.
- [29] K. Beznosov and Y. Deng, "A Framework for Implementing Role-based Access Control Using CORBA Security Service," In *Proceedings of 4^d ACM Workshop on Role-Based Access*, October, 1999.
- [30] Bob Blakley, "The Emperor's Old Armor," In *Proceedings of the 1996 ACM New Security Paradigm Workshop*, 1996, pp. 2-16, ACM.