Integrating SSL to the *JaCoWeb Security* Framework: Project and Implementation

M. S. Wangham, L. C. Lung, C. M. Westphall, J. S. Fraga
Federal University of Santa Catarina – LCMI-DAS-UFSC
Campus Universitário – Trindade — Florianópolis – SC – Brazil
PO Box 476 - CEP 88040-900
{wangham, lau, merkle, fraga}@lcmi.ufsc.br

Abstract

This paper presents an integration of SSL and JacORB, according to the CORBA security model, which does not affect the functionality and characteristics of the ORB. The project and implementation of this framework take flexibility and interoperability concerns related to the integration of security technologies in consideration, aspects that are not well described in OMG CORBAsec specifications. Therefore, the main requirements of open systems are assured: portability, reusability and interoperability.

Keywords

CORBA, SSL, security in distributed systems, CORBAsec.

1. INTRODUCTION

Large-scale networks are characterized by the great space distribution, with an open feature, integrating significant amounts of computational resources designated by its heterogeneity. A recent concern in relation to these new applications is the definition of new services that observe the security requirements in large-scale networks. However, it presents several difficulties, as there is a large number of users, objects and operations, along with the lack of a security policy enforcement and heterogeneous environments, present in large-scale networks. Thus, complexity and scaling increase the difficulty of security policy management.

Also, the advent of new paradigms and distributed-systems programming tools, such as the open-system distributed objects paradigm, implemented with the use tools like Java, CORBA (*Common Object Request Broker Architecture*), and Web, introduced new security concerns.

Many of these distributed applications follow the object-oriented paradigm and consider CORBA [1] the best alternative for adjusting to open systems requirements. As a result of this, the OMG (*Object Management Group*) introduced the CORBA Security Service specifications (CORBASec) [2], which, if implemented and managed properly, can provide a high level of security in large-scale environments. The CORBASec specifications, which still must pass for some extensions (and updates), define a useful set of service interfaces and facilities for implementation of secure applications in heterogeneous distributed systems.

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 code at any point of the network.

Among the various distributed system technologies available, Netscape's SSL (*Secure Socket Layer*) stands out for being a cryptographic protocol widely used in Internet applications. SSL is an adequate general-purpose protocol for connection-protection in distributed systems, and for assuring authenticity, privacy and integrity in communications through TCP/IP connections.

The *JaCoWeb Security* project (*http://www.lcmi.ufsc.br/jacoweb*), has been developed in our laboratories, aims at integrating the CORBA security model with Web and Java security models to compose an authorization scheme for distributed applications in large-scale networks [3 and 4]. This scheme has been developed as an attempt to implement global policies, certainly a great challenge, and even more in large-scale distributed systems such as the Internet. In this environment, distributed applications are expressed in the form of clients represented by Java applets, and of servers, available via CORBA application objects.

The implementation of an authorization scheme does not depend only on access control. Other internal controls are also important for the implementation of authorization policies, such as cryptographic controls, authentication and identification services etc. In this work, to implement these cryptographic controls, the SSL was used as the underlying technology. The SSL protocol was chosen for this work for having been inserted, by the OMG, in its last security-services revision [2]. However, the 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*), without compromising interoperability.

The *JaCoWeb Security* framework integrates ORB and SSL, based on the service objects approach. It will not alter ORB's original characteristics and functionality, thus allowing ORB+SSL to perform both conventional and secure connections. This framework is defined according to the CORBA security specifications [2].

Initially, the paper presents the security model of the CORBA standard in section 2. In section 3, security technologies are introduced. The *JaCoWeb Security* framework is proposed in section 4, while section 5 discusses implementation aspects. In section 6, the framework's performance is analyzed. Section 7 brings some general remarks and section 8 concludes the work.

2. SECURITY IN THE CORBA/OMG SPECIFICATIONS

According to CORBA architecture, proposed by the OMG, it is possible to have remote access to object methods through an ORB - in heterogeneous distributed environments and in a transparent way. The ORB, in a general sense, is a communication channel for distributed objects. In CORBA, interoperability among objects is obtained through the objects' IDL (*Interface Definition Language*) specifications. In object-oriented applications development, the CORBA standard offers an additional COSS (*Common Object Service Specification*) [5] that provides an easier application development. These services are OMG standards within the OMA (*Object Management Architecture*) perspective. These service objects form a collection of services (interfaces) at the system level, and offer basic functionality for usage and implementation of application objects. The COSS specifications can be seen as extensions/complements to the ORB functionality.

When a client invokes a method in a remote server, there are a large number of actions to be performed on the requisition, so that it may, through the communications' channel, reach the server. A client request becomes a *Request* object, containing a number of attributes, which define the target object (the server), the requested operation (the method), the invocation arguments, the reply, in addition to a set of methods that allow manipulating these attributes. Once created by the ORB, responding to a client request, the *Request* object goes through *marshalling* and is converted to a byte stream, so it can be transmitted via the transport layer. In the CORBA standard, messages exchanged among the objects are formatted according to the GIOP protocol (*General Inter-ORB Protocol*) and transmitted using the IIOP protocol (*Internet Inter-ORB Protocol*), which uses TCP/IP for transport (GIOP + TCP/IP = IIOP).

2.1. CORBAsec

The OMG issued a document defining the main concerns regarding distributed object security [2]. The security model proposed in the document establishes some procedures regarding authentication and authorization verification in remote method calls, secure communications among objects, besides aspects concerning delegation of rights, non-repudiation, audit, and security administration.

The CORBA security model relates objects and components in four levels of a system (figure 1): the application level (clients and servers); service objects, ORB services, and the ORB core (all at the middleware level); security technology components (underlying security services level); and basic protection components (a combination of hardware and local operating systems).

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. In the specifications of the CORBA security model, the ORB services are implemented with *interceptors*. An interceptor is interposed in the path of an invocation between a client and a target object. Each COSS service related to security is associated with an interceptor, whose purpose is to cause the transparent deviation to the corresponding service.

In the CORBA security model, two types of interceptors are defined and act during a method invocation: the *Access Control Interceptor*, at a higher level, is invoked to perform access control tasks on the call, and the *Secure Invocation Interceptor*, at a lower-level, provides integrity and confidentiality properties in the corresponding invocation exchanges. These interceptors act both on the client and on the server application objects.

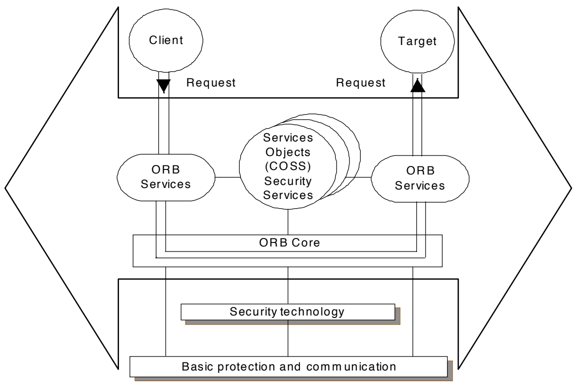


Figure 1: CORBA Security model

Inside the arrows, in figure 2, are the access control interceptor (represented by the *Access Control* object) and the secure invocation interceptor, represented by the *Secure Invocation* object. The service objects that implement the security control in CORBA specifications are: *PrincipalAuthenticator*, *Vault*, *Credential*, *DomainAccessPolicy*, *RequiredRights* and *AccessDecision*. The *PrincipalAuthenticator* object corresponds to the principal-authentication service. The *Vault* object establishes the secure associations between clients and servers with the respective security contexts, and the *SecurityContext* object represents the secure association context. The *Credential* object represents the client rights and credentials in the session. The *DomainAcccessPolicy* 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 [6]. The *RequiredRights* object stores information on the necessary rights for execution of each method in each system interface. The *AccessDecision* objects are responsible for interacting with *DomainAccessPolicy* and *RequiredRights*, to determine if a given operation on a specific target object is allowed. There are other service objects that deal with non-repudiation and audit.

The *Secure Invocation* object is responsible, at bind time, for the establishment of a secure association between the client and the server. At bind 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 the client and the server objects. This *SecurityContext* object provides the security context information and is used by the *Secure Invocation* object to protect messages, to maintain integrity and/or confidentiality.

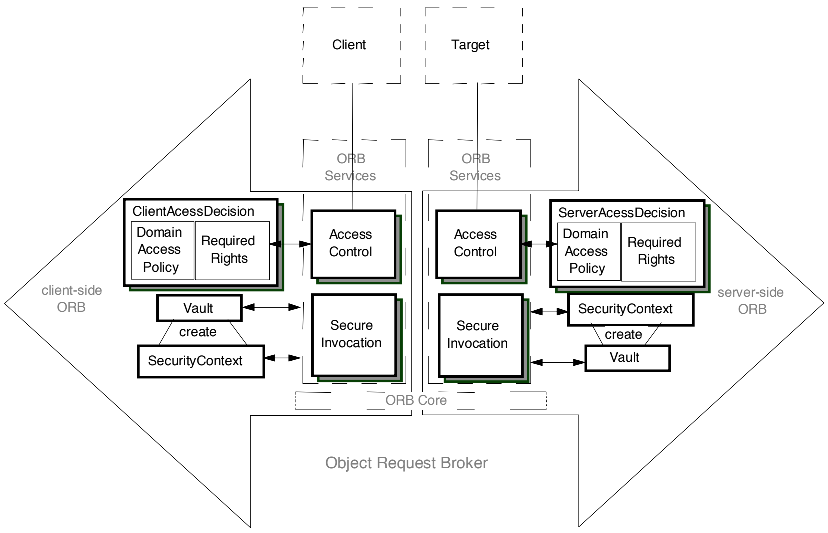


Figure 2: ORB security services

3. SECURITY TECHNOLOGIES

The service objects in the CORBA security model isolate applications and the ORB from the security technology (as in figure 1), which consists of an underlying layer that implements some functionality of related security service objects. The security technology includes services such as authentication, secure association (certificates, encryption and decryption), and others. According to the CORBAsec specifications, technologies that may be used to provide these services are [2]: SPKM (*Simple Public Key GSS-API Mechanism*), Kerberos, CSI-ECMA (based on SESAME and ECMA GSS-API) and SSL.

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

CORBAsec was initially developed for static applications in restricted environments and cannot be easily adapted to new requirements and trust relationships of Internet-based applications. The OMG firewall draft and the integration of SSL into CORBA are the first attempts to bring CORBAsec to the Internet [7].

3.1. The SSL protocol

The SSL [8] provides authenticity, confidentiality, and integrity for communications through TCP/IP connections. This protocol is located between the transport-layer protocol (for example, the TCP) and the application-layer protocol, and consists of two layers: the **SSL Record** and the **SSL Handshake** protocols. The SSL Record layer provides secure connections, assuring integrity and confidentiality through symmetric cryptography and integrity messages (MAC -

message authentication code). The SSL Handshake layer allows client and server authentication, negotiates cryptographic algorithms, and generates the symmetric key to be used by the SSL Record.

The SSL uses digital certificates and digital-signature transfers for client and server authentication. The SSL session is established when the initial handshake between client and server takes place.

SSL authentication uses X.509 certificates standard to transfer an applications' public key information. The X.509 certificates have certifying authorities all over the world. It is an international standard appropriate to and widely accepted in a wide variety of fields.

4. JaCoWeB AUTHORIZATION SCHEME: ORB+SSL

The *JaCoWeb* authorization scheme (*http://www.lcmi.ufsc.br/jacoweb/*) corresponds to an access control structure based on two levels of control: a global and a local level, in the application object hosts. The *PoliCap* policy service aims to provide a policy object management service centralized in a domain of distributed object applications. It establishes the first level of verification, at binding time. The second access control level based on capability mechanisms is carried through in application execution time, reflecting the *PoliCap* policies, allowing verifications on both sides – on the client and server sides – in method invocations. The capability mechanism is set up using abstractions of CORBA itself. These two levels of control are performed by the access control interceptor of CORBAsec model. More details about the policy object management service and the capability mechanism are present in [4 and 9].

The CORBAsec secure invocation interceptor tasks, such as the establishment of a secure association between the client and the server, and the protection of messages, are the main topics addressed in this work and are essential to the access control interceptor tasks proper functioning. The framework that deals with this tasks was called *JaCoWeb Security Framework*.

The definitions for the proposed framework, integrating ORB and SSL support, started out with the study and detailed analysis of the structure of a conventional ORB. We chose a free Java ORB to be used in our framework - the JacORB [10]. This middleware is basically a combination of development tools and a support for building and integrating open-system object-oriented applications. The JacORB was entirely written in Java, according to the OMG specifications.

In figure 3, the JacORB project implementation model is detailed [10]. The flow of the invocation within the ORB, starting from the moment the request-data (already in byte stream) reaches the server's socket is shown. The *BasicAdapter* module, through a *Connection* object, deals with the low-level functions handling the TCP/IP connection between the server and the client (the *ServerSocket* objects). The *RequestReceptor* object performs *unmarshalling* on the byte stream to rebuild the *Request* object on the server side. The *Request* is then turned into a *ServerRequest* object, ready to be sent to the *RequestQueue* and to the POA

(*Portable Object Adapter*). The main functions of the POA are generation and interpretation of object references, method invocation, object and implementation activation and deactivation, etc.

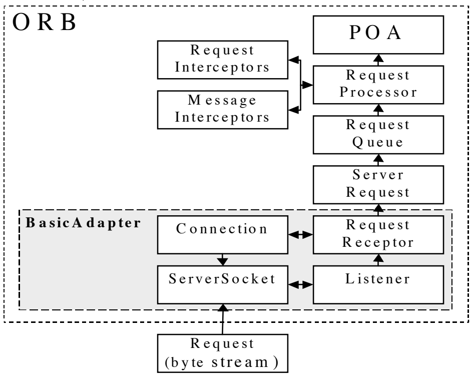


Figure 3: Server-side Data Flow

Two ways of integrating SSL technology and ORB were discussed during the project phase. In this work, we attempted to define a solution that would allow the ORB to deal with both secure and conventional connections, thus maintaining the ORB's original functionality. The first alternative consists in implementing the entire SSL specifications as a part of the security service (figure 2). In this case, the model is completely respected, and security concepts are created and maintained by the security service. In other words, the message interceptor has the complete control over the handshake process (item 3.1). The major problem with this implementation is its high cost, since it would be necessary to reorganize an available SSL code, in order to have an API to be accessed by the *Vault* object. The other solution consists in using an SSL package with available executable code, and model it according to the needs. In this case, however, some of the functionality of the message interceptor in controlling the handshake process is lost. The second alternative was chosen, since, not having to implement the entire SSL specifications, the *JaCoWeb* team could pay special attention to other CORBA security model services [6].

4.1. Integrating SSL to the ORB

The framework for integrating ORB/CORBA and SSL support consists in encapsulating an available SSL package in the form of a service object, such as the COSS objects. The iSaSiLk [11] package was chosen because it has presented all the functionality required for the proposed framework. The objects, with IDL-defined interfaces, which compose the JaCoWeb Security package for the proposed integration, are shown in figure 4. The *Security* class encapsulates all functionality of the iSaSiLk package and is instantiated by the application at creation-time. The parameters defined at creation of the *Security* object allow iSaSiLk to define the *SSL Context* (figure 4). This *SSL Context* defines how the client and the server should choose the *Cipher Suite* (set of cryptographic algorithms).

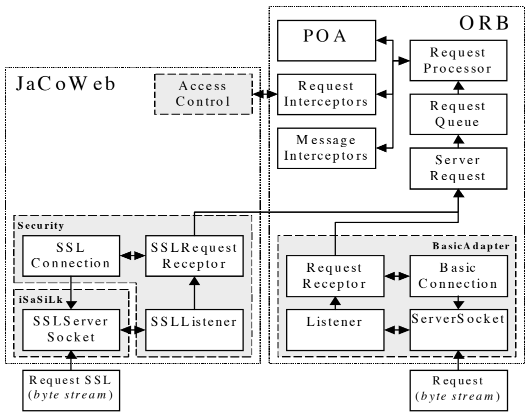


Figure 4: The ORB+SSL *framework*.

Also in figure 4, the *ServerSocket* and the *SSLServerSocket* are shown: one for establishing normal connections between the *Listener*, and one for establishing secure connections through the *SSLListener*. The *SSLServerSocket* is provided by iSaSiLk and is encapsulated by JaCoWeb. The *SSLConnection* object instantiates the *SSLSocket* for establishment of a secure connection between client and server. So, the secure call is established inside the JaCoWeb package and, as a result, the requisition byte stream is passed on to the ORB through the *SSLRequestReceptor*. The request interceptor must activate the security service to perform access control tests on a method call [3].

The key element for this integration to work properly is the definition of the way an application selects the connection mode to be used (secure or conventional). In the CORBA model, an object can be located in the distributed system through its object references. In the OMG, these object references are defined by the IOR specification (*Interoperable Object Reference*). The IOR [5] is a character sequence which, when converted to proper formats, supplies information such as the port, the IP address, a pointer to the object to be accessed and some IOR-specific control data. The CORBASec specification defines an IOR extension in order to be able to gather specific information for the ORB, so the ORB becomes able to handle secure connections. This IOR extension is done by adding a security tag to the body of the IIOP's IOR profile.

5. JaCoWeb SECURITY IMPLEMENTATION

In this section, the ORB+SSL implementation is addressed. This implementation uses the CORBAsec specifications and was developed on a JacORB beta 13 platform [10].

5.1. The framework's IDL interface

In this work, Web applications are defined as clients represented by Java signed applets, and CORBA servers (remote objects) interacting through a secure

ORB. The example developed in the JaCoWeb Security implementation is an Internet banking-system prototype. When applications establish secure connections using the proposed framework, some security data are initialized. The *Security* object is responsible for initializing security using the IDL in figure 5. After the ORB and POA are initialized (only for servers), the applications complete the secure ORB initialization by instantiating a *Security* object according to the application type (stand-alone/applet client or server).

For Web applications, security information is defined in the first method on the IDL interface (figure 5: `initAsClient(orb, applet)`). This method instantiates a *Security* object and invokes the `initAsClient (orb)` method.

For stand-alone applications, the security context is defined in the `initAsClient (orb)` method, which must define the *principal* according to the client's certificate, creates the client's static SSL credential (based on this client's certificate), creates and defines configurations to be used in the establishment of the secure connection, defines the client's SSL context, specifying cipher suites, the *TrustDecider* object (in charge of server authentication) and the client's certificate, and finally, defines these security configurations (*Security* object) in the `jacorb.orb. Environment` class.

```
// $Id: Security.idl, v1.0 1999-12-08
#include "orb.idl"
#pragma prefix "edu.lcmi"
module jacoweb {
interface Security {
Security initAsClient (
in ORB        orb,
in Applet     applet );
Security initAsClient (
in ORB        orb, );
Security initAsServer (
in ORB        orb,
in PortableServer::POA poa);};
};
```

Figure 5: IDL interface of *JaCoWeb* package

The security tag with the SSL connection definitions is defined in the SSLIOP (*Secure Socket Layer Inter-ORB Protocol*) module [2]. The size increase of the IOR does not cause any additional compatibility concerns, since the IOR's structure is flexible enough, and the Naming Service already accepts diverse sizes.

To establish a secure connection using SSL, the application server must register itself in the Naming Service using an IOR containing an SSL Tag. In order to allow these modifications in the ORB.createIOR( ) method of JacORB are needed. Since clients need to know the SSL port and the requirements for establishing a secure connection, the ParsedIOR class, responsible for IOR decoding, was also altered.

6. JaCoWeb SECURITY EVALUATION

The *JaCoWeb* authorization scheme, using discretionary access control policies, was evaluated according to the Common security evaluation Criteria [4]. The *Common Criteria* (CC) is the result of a joint effort of various international companies attempting to develop a unique security evaluation criteria for distributed-system information-technology products and systems. The CC defines how to express system or product behavior (functional requirements), and whether or not the product/system is effectively and properly implemented. It determines if the product or system fulfills its assurance requirements, so users can trust the system or the product's security.

The threats our projects aims at handling were defined as T.ACCESS, T.CAPTURE, T.INTEGRITY, T.SECRET, and T.IMPERSON. A T.ACCESS threat occurs when a user obtains information or system resources without the proper authorization. T.CAPTURE happens if an intruder attempts to modify or obtain system information by eavesdropping, while T.INTEGRITY takes place when transmitted information may be modified due to user or transmission errors. In a T.SECRET threat, a user can take, either intentionally or by accident, unauthorized information, and finally, T.IMPERSON happens when an intruder can get access to information or resources impersonating an authorized user. Some threats could not be handled in the project, as the user abuse, when an user requests excessive system resources, and, consequently, causes denial of service to another authorized user. Also, there is not a way to penalize a user for eventual harmful behavior, since audit services were not implemented.

The evaluation of *JaCoWeb Security* reaches the CC EAL3 level. In other words, the project was fully and methodically tested and verified, and is appropriate in cases where moderate security is demanded. The main focus in this paper was to integrate SSL and ORB, and to evaluate its performance. This performance analysis completes the qualitative analysis done in [4].

6.1. JaCoWeb Security Performance

The SSL performance is very important, as it is the core capability of CORBAsec level 2 products that are currently emerging into the market place [12]. In order to evaluate the additional cost for integrating security mechanisms like SSL to the ORB, as in [12], we present the performance analysis of the proposed *JaCoWeb Security* framework. The measurements were carried out with one client and one server. We compare the measurements obtained in a conventional ORB (JacORB) with those obtained using *JaCoWeb Security*.

The first measure used for evaluation was latency. In *JaCoWeb*, when the client executes the first request to a server, a number of actions takes place to establish the SSL security context. The public-key algorithm used for mutual authentication and key-distribution was the 1024-bit RSA. Data ciphering used the symmetric block-ciphering 3DES_EDE and the SHA *hash* security function for MAC computation.

The two tables in figure 6 (a) show the average time, in milliseconds, of 1000 *pings*[1] between client and server. The first table shows measurements taken

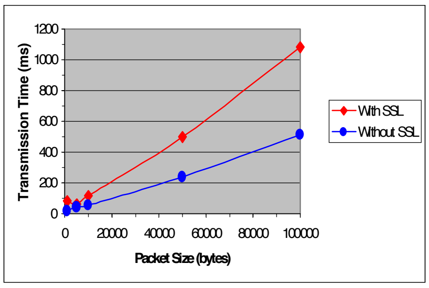
[1] Ping (Packet INternet Grouper) - is a tool which, among other things, can be used to test connection quality or reply time of a call.

on a 300 MHz Pentium III machine running both client and server (standalone). This measure allows calculation of additional processing time required by the SSL.

Standalone	Time (ms)
JaCoWeb (first call)	1600
JacORB (first call)	4
JaCoWeb (average of the next 1000 calls)	8
JacORB (average of the next 1000 calls)	4

Rede Local (Ethernet)	Time (ms)
JaCoWeb (first call)	1700
JacORB (first call)	5
JaCoWeb (average of the next 1000 calls)	10
JacORB (average of the next 1000 calls)	5

(a)



(b)

Figure 6: Comparing the latency of *JaCoWeb*

The second table lists the measurements carried out on a local network (100 Mbps Ethernet), with the client running on a 300 MHz Pentium III and the server on a 400 MHz Pentium II. In this case, besides the additional processing time required by the SSL, the relative cost of network interactions for creating the SSL context is accounted for. Noticeably, the first call on the *JaCoWeb* is relatively high (approximately 1600 ms processing time plus 100 ms for SSL context establishing, totaling 1700 ms). This is because computational processing required for encryption and decryption of public keys, which is needed for initializing the first SSL connection. However, for the subsequent pings, the approximate time of 10 ms is acceptable.

The other measure used for evaluating the *JaCoWeb Security* was packet transmission time (figure 6 (b)), for varying packet sizes, between a client and a server running on the same machine. The packet sizes used were 1, 5, 10, 50 and 1100 Kbytes. It can be seen that the transmission times for the *JaCoWeb* are roughly twice the time for the conventional ORB (JacORB) used. These results show the increase in cost since additional security support is needed for ciphering of messages over a network.

7. GENERAL REMARKS

In this section, the ORB+SSL integration is evaluated according to the CORBASec specification [2] requirements and further specification considerations in [13]. The requirements on which this evaluation is based are transparency, simplicity, reusability, scalability, flexibility, and interoperability.

Transparency hides the security system's functionality from the end-user and from the application developer, as if the invocations were treated locally. Simplicity requires the security system to be easy to understand, use, and manage. *JaCoWeb Security* tries to respect these two requirements by defining the *Security* object (section 5.1), which will be responsible for initializing and activating all the security mechanisms. From then on, all calls are carried out as in a conventional

ORB. In *JaCoWeb Security* implementation the system configurations are defined statically, at compile-time. However, the security policies for application access control can be dynamically defined at run-time through a management interface.

Scalability concerns the adequacy of the security system to both large and small systems, in number of users and operations. Therefore it must have support for domain policies, groups, roles, and so on. Our implementation uses group and role concepts inside a single domain policy, in a simplified way. *JaCoWeb Security* project envisions the possible use of LDAP (*Lightweight Directory Access Protocol*), an implementation of directory services based on the X.500, sufficiently used nowadays [14], and the use of a CORBA PKI (*Public Key Infrastructure*)[2], to make feasible its use in large-scale networks. Each domain would be composed by a local LDAP server, connected logically to other servers forming the global name space, that would be responsible for storing CORBA objects references. The CORBA PKI would deal with the X.509 certificates used in the CORBA environment.

In [13], the relations between flexibility and interoperability are discussed. The highest degree of flexibility is attained when the number of standardized interfaces and protocols is as small as possible, allowing developers and users to extend the security mechanisms according to their needs, in an easier way. In opposition, maximal interoperability is obtained when there is a large number of standardized interfaces and protocols, thus making sure ORBs from different vendors offer the same functionality, allowing and encouraging applications in different ORBs to interact.

It is difficult to completely fulfill each requisite without affecting the others. *JaCoWeb Security* uses the options offered by CORBASec to balance these requirements. Using IDL interfaces and the service-object model in the implementation of *JaCoWeb Security*, the interoperability of the system was maintained. However, since the CORBASec specifications standardize a great deal of security abstractions (mechanisms, properties, policies, data structures, and so on), the system's flexibility is greatly impaired. Although proprietary extensions on the model can be implemented as service objects without altering the ORB's characteristics.

Regarding communications, to establish security contexts through underlying security technology, *JaCoWeb Security* uses the SSL with the SSLIOP. This means interoperability with other ORBs is only achieved when both use the same protocol (SSLIOP) and have "consistent" security policies [13]. This excludes ORBs using the SECIOP (*Secure Inter-ORB Protocol*) adaptation protocol (for SPKM, Kerberos and CSI-ECMA – item 3) and DCEIOP (for DCE). Nevertheless, the *JaCoWeb Security Framework* was modeled to allow an easy adaptation to other security technologies, they must have an available API.

An academic attempt to implement CORBA security models in Java can be found in *Nephilim: Java Implementation of CORBA Security Services* [15], using **CSI-ECMA** as security technology. The *JacORBoverSSL* project, carried out by

[2] ftp://ftp.omg.org/puc/docs/ec/99-12-03.pdf

Andre Benvenutti, is also a Java implementation that tries to integrate an ORB and the SSL. However, the *JacORBoverSSL* does not conform to the CORBA security model levels, because the implementation was done inside the ORB core. The JacORB is already using *JacORBoverSSL*, placed in the ORB core, in its late release.

Besides these prototypes, there are few commercially available products integrating the ORB and security technology, such as ORBAsecSL2 [16], OrbixSecurity [17], and SecureBroker SSL [18]. **OrbixSecurity** guarantees communications security through the use of SSL. However, this implementation follows the CORBASec-defined interfaces, without implementing standard interceptors, and uses filters and transformers as deviation mechanisms. According to [16] and [18], **ORBAsecSL2** and **SecureBroker** follow the CORBASec model, using Kerberos and SSL as the security technology. SecureBroker is still in the development stage and the only available part is the SSL integration. These SSL product implementations do not use the **OMG's Service Object approach** [5] to integrate SSL and the ORB. Owing to this fact, interoperability problems could easily arise. Otherwise, our work was totally developed as an OMG Service Object, composed by an IDL interface and operations. This approach allows the communication among distinct ORBs with the same security technology.

8. CONCLUDING REMARKS

The objective of *JaCoWeb Security* project is to develop a security scheme for distributed applications in large-scale networks. Solutions for adding security mechanisms to the ORB were discussed. A general framework was proposed for integrating security technology to the ORB without compromising the ORB’s interoperability. The proposed framework adapts a security mechanism, as an API, calling the security mechanisms of an underlying security service.

Using SSL as an underlying security technology guarantees integrity, confidentiality, and authenticity in critical Web applications. The CORBA platform can be used for integrating systems and applications, supplying the flexibility required by such like Web dynamically changing business environments. For these applications it is further necessary to guarantee resources will not be used by unauthorized people or in an unauthorized manner. In this case, authorization mechanisms, which also compose the CORBASec model, can also be used. In [3 and 9], the proposed *JaCoWeb Security* authorization scheme is explained in depth.

Besides, all the *JaCoWeb Security* framework implementations used for performance tests were fully developed. These evaluations reveal the additional computational cost required for guaranteeing secure ORB connections using SSL.

ACKNOWLEDGMENT

We would like to thank Gerald Brose, author of the JacORB [10], and André Benvenutti for their kind attention helping us to solve our main difficulties in the beginning of this work.

REFERENCES

[1] OMG, “The Common Object Request Broker 2.0/IIOP Specification”, 1996.

[2] OMG, “Security Service: v1.2 Final”, Nov. 1998 .

[3] C. M. Westphall and J. S. Fraga, “Authorization Schemes for Large-Scale Systems based on Java, CORBA and Web Security Models”, The IEEE ICON’99, pp. 327-334, Sep. 1999, Brisbane-Queensland, Australia.

[4] C. M. Westphall, J. S. Fraga and M.S. Wangham, “Policap – Policy Service for CORBA Security model”, XVIII SBRC, 23-26 de Maio, pp. 355-370, Minas Gerais – Brasil, 2000 (in portuguese).

[5] OMG, “CORBAservices: Common Object Services Specification”, 1997.

[6] G. Karjoth, “Authorization in CORBA Security” In *Proceedings of the Fifth ESORICS*, LNCS, pp. 143-158, Springer-Verlag, Berlin Germany, Sep. 1998.

[7] OMG, "Joint Rev. CORBA/Firewall Security," Doc.orbos/98-05-04, Jun. 1998.

[8] A. O. Freier, P. Karlton and P.C. Kocher, “Secure Socket Layer 3.0”, Internet Draft, Nov. 1996.

[9] C. M. Westphall, “An Authorization Scheme for Security in Large Scale Distributed Systems”, PhD Thesis, EEL Depart., UFSC, Dec. 2000 (in portuguese).

[10] G. Brose, “JacORB: Implementation and Design of a Java ORB”, Procs. Of DAIS’97, IFIP WG 6.1, Cottbus, Germany, Sep. 1997.

[11] IAIK-Java Group, “iSaSiLk 2.5 User Manual”, Inst. for Applied Information Processing and Communications, Graz, Univ. of Technology, Nov. 1999.

[12] T. M. Blaser, “Information Encryption Prototype for testing SSL Performance”, NASA, ppt slides, OMG DOCsec Workshop, April, 2000.

[13] U. Lang and R. Schreiner, “Flexibility and Interoperability in CORBA Security”, Elet. Notes in Theoretical CS Vol. 32, Elsevier Science, 2000.

[14] Johner, H., et al. "Understanding LDAP," SG24-4986-00.

[15] “Nephilim : Java Implementation of CORBA Security Services”. *University of Illinois*. http://choices.cs.uiuc.edu/Security/nephilim.

[16] Adiron, LLC., “ORBaSecSL2 – User Guide”, version 2.0, April 1999.

[17] Iona Technologies, “Orbix Security 3.0 – White Paper”, September 1999.

[18] Promia, “SecureBroker”. http://www.promia.com/products/securebroker.html