Projeto, Implementação e Avaliação do *JaCoWeb Security*

Michelle Silva Wangham, Lau Cheuk Lung, Carla Merkle Westphall,
Joni da Silva Fraga, Ricardo Sangoi Padilha, Luciana Moreira Sá de Souza
Laboratório de Controle e Microinformática (LCMI) – DAS – CTC – UFSC
Campus Universitário – Caixa Postal 476 – Trindade - CEP 88040-900 – Florianópolis – SC – Brasil
e-mail: {wangham, lau, merkle, fraga, padilha, luciana}@lcmi.ufsc.br

Resumo

Uma crescente demanda de serviços e novas aplicações têm provocado o surgimento de novos paradigmas e ferramentas de programação distribuída. Java, Corba e Web fazem parte dessas novas tecnologias. As especificações da OMG não apresentam uma visão clara sobre a integração do ORB com as tecnologias de segurança sem comprometer a portabilidade e interoperabilidade do ORB como um todo. Este artigo propõe um *framework* de integração (ORB+SSL) que não altera as funcionalidades e as características originais do ORB, assegurando assim os principais requisitos de sistemas abertos: portabilidade, reusabilidade e interoperabilidade.

Palavras-chave: *CORBA, SSL, segurança em sistemas distribuídos, sistemas abertos*

Abstract

A growing demand of services and new applications has caused the appearance of new paradigms and distributed programming tools. Java, Corba and Web are part of those new technologies. The OMG specifications do not present a very clear vision about structuring the ORB to integrate security technologies without affecting the portability and interoperability of ORB. This paper proposes an integration framework (ORB+SSL) that does not alter the functionalities and characteristics of a conventional ORB. Therefore, the main requirements of open systems: portability, reusability and interoperability are assured.

Keywords: *CORBA, SSL, security in distributed systems, open systems*

1. Introdução

A segurança em sistemas distribuídos de larga escala, tal como a Internet, tem sido uma área de pesquisa que tem atraído bastante interesse da comunidade científica. Aplicações críticas, tais como sistemas de informações gerais, sistemas bancários, comércio eletrônico, dentre outros, devem contar com mecanismos de segurança para garantir a integridade, confidencialidade e autenticidade para que possam operar de maneira correta, eficaz e segura.

O surgimento de novos paradigmas e ferramentas de programação em sistemas distribuídos como o paradigma dos objetos distribuídos em sistemas abertos, implementado pelo uso de ferramentas como Java, CORBA (*Common Object Request Broker Archtecture*) e Web, introduziram novos problemas de segurança, implicando na necessidade de novos modelos e conceitos. Sistemas de objetos distribuídos são constituídos por um grande número de camadas de *software* para serem implementados, como por exemplo, a ferramenta de programação, *middleware*, serviços de segurança, tecnologia de segurança, sistema operacional, rede, dentre outras que possam existir [8]. Vulnerabilidades podem ocorrer entre cada uma dessas camadas, pois a interação entre as mesmas é bastante complexa. A complexidade dos vários níveis é ainda mais complicada em sistemas onde a garantia da segurança é amplamente distribuída.

Dentre as várias tecnologias de segurança em sistemas distribuídos, o SSL (*Secure Socket Layer*), desenvolvido pela *Netscape*, se destaca por ser um protocolo criptográfico amplamente utilizado em aplicações na Internet. O SSL é um protocolo de propósito geral adequado para proteger conexões em sistemas distribuídos, prover autenticação, confidencialidade e integridade para comunicações através de conexões TCP/IP. Este protocolo é utilizado neste trabalho por estar inserido pela OMG[1] (*Object Management Group*), em sua última revisão do serviço de segurança [15], como um dos protocolos a serem utilizados por aplicações que implementam a segurança no CORBA.

[1] É uma organização formada por mais de 700 empresas com o objetivo de especificar um conjunto de padrões e conceitos para a programação orientada a objetos em ambientes distribuídos abertos.

Uma plataforma CORBA [13] é adotada neste trabalho no sentido de atender aos requisitos de sistemas abertos. As especificações do CORBA formam um conjunto de padrões e conceitos para objetos distribuídos, propostos pela OMG. Dentre as várias especificações COSS (*Common Object Service Specificaction*) [14] aparece o serviço de segurança que define um conjunto de mecanismos para autenticação, autorização, auditoria, segurança da comunicação e não-repudiação.

O *JaCoWeb Security* tem como objetivo o uso do modelo CORBA de segurança integrado com os modelos de segurança Java e Web de modo a compor um esquema de autorização para aplicações distribuídas em redes de larga escala. Esse esquema está sendo desenvolvido com o objetivo de concretizar a implantação de políticas globais, o que representa um grande desafio, principalmente em redes de larga escala como a Internet. Como um esquema de autorização é a concretização da política de autorização através de um conjunto de mecanismos, a implantação deste esquema não depende só do controle de acesso, outros controles internos, tais como, controles criptográficos, serviços de autenticação e serviços de identificação, são também importantes na concretização da política de autorização.

Para implementar estes controles criptográficos, o protocolo SSL foi utilizado neste trabalho como tecnologia de segurança subjacente. No entanto, constata-se que a especificação CORBAsec não apresentam uma visão clara, em termos de conceitos e necessidades, sobre a integração do ORB com as tecnologias de segurança sem comprometer o aspecto da interoperabilidade. É proposto neste artigo um *framework JaCoWeb Security* que integra, baseado na abordagem de objetos de serviço, o ORB com a tecnologia de segurança SSL, sem alterar as funcionalidades e as características originais do ORB, permitindo assim que o ORB+SSL faça tanto conexões convencionais como seguras. Este *framework* é definido segundo as especificações de segurança do CORBA [15].

Este artigo está dividido da seguinte maneira: na seção 2 tem-se o serviço de segurança da OMG; na seção 3, são abordadas as tecnologias de segurança e o protocolo SSL; na seção 4, o *framework JaCoWeb Security* é apresentado; na seção 5, são discutidos aspectos da implementação; na seção 6, o desempenho do *framework* implementado é analisado; na seção 7, são apresentadas as considerações gerais e trabalhos relacionados; e finalmente, na seção 8, as conclusões do trabalho.

2. Serviço de segurança no padrão CORBA

As especificações CORBA correspondem a um conjunto de padrões e conceitos para objetos distribuídos em ambientes abertos, propostos pela OMG (*Object Management Group*). Segundo essa arquitetura, métodos de objetos remotos podem ser invocados de forma transparente em ambientes distribuídos e heterogêneos através de um ORB (*Object Request Broker*). O ORB, num sentido mais genérico, é um canal de comunicação para objetos distribuídos. No CORBA, a interoperabilidade entre objetos é conseguida a partir das especificações IDL (*Interface Definition Language*) das interfaces de objetos. No desenvolvimento de aplicações orientadas a objetos, o padrão CORBA oferece ainda um conjunto de objetos de serviço (*COSS*) [14] que facilitam o desenvolvimento da aplicação. Esses serviços são padronizados pela OMG dentro da perspectiva da arquitetura OMA (*Object Management Architecture*). Os objetos de serviço formam uma coleção de serviços (interfaces) em nível de sistema que oferecem funções básicas para o uso e a implementação de objetos de aplicação. As especificações COSS podem ser entendidas como extensões ou complementos das funcionalidades do ORB.

Quando um cliente faz uma invocação de método em um objeto servidor remoto, há todo um conjunto de procedimentos no tratamento da requisição para que esta possa ser transmitida no canal de comunicação até chegar no servidor. Uma requisição do cliente é transformada em um objeto *Request*, o qual contém um conjunto de atributos que define o objeto destino (o servidor), a operação requisitada (o método), os argumentos da invocação, a resposta, entre outros; e um conjunto de métodos que permitem manipular esses atributos. Uma vez criado pelo ORB, a partir de uma requisição do cliente, o objeto *Request* passa pelo processo de conversão (*marshalling*) para *byte stream* para que possa ser enviado pela camada de transporte. No padrão CORBA, as mensagens trocadas entre os objetos são formatadas de acordo com o protocolo GIOP (*General Inter-ORB Protocol*) e transmitidas via protocolo IIOP (*Internet Inter-ORB Protocol*), o qual efetivamente usa o TCP/IP como meio de transporte (GIOP+TCP/IP = IIOP).

2.1 Descrição do serviço de segurança CORBA (CORBA Security)

A OMG redigiu um documento definindo as principais linhas as quais se referem à segurança de objetos distribuídos [15]. O modelo de segurança descrito neste documento estabelece vários procedimentos envolvendo a autenticação e a verificação da autorização na invocação de um método remoto, a segurança da comunicação entre os objetos, além de aspectos relacionados com esquemas de delegação de direitos, não-repudiação, auditoria e administração da segurança.

O modelo CORBA de segurança relaciona objetos e componentes de quatro níveis de um sistema (figura 1): nível de aplicação (cliente e servidor); objetos de serviço, serviços ORB e o núcleo do ORB (todos em nível de *middleware*); componentes de tecnologia de segurança (em nível de serviços de segurança subjacentes) e componentes de proteção básica, fornecidos por uma combinação de hardware e sistemas operacionais locais.

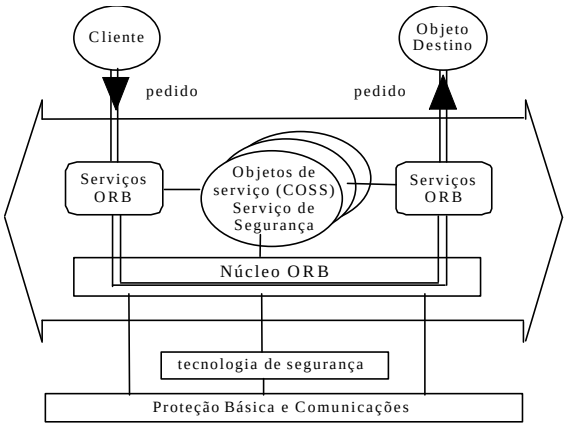


Figura 1. Modelo CORBA de segurança.

No modelo CORBA de segurança, são definidos dois tipos de interceptadores[2] que atuam durante uma invocação de método: o interceptador de controle de acesso (*Access Control Interceptor*) que em nível mais alto provoca um desvio para realizar o controle de acesso na chamada, e, o interceptador de chamada segura (*Secure Invocation Interceptor*) que faz uma interceptação de mais baixo nível no sentido de fornecer propriedades de integridade e de confidencialidade nas transferências de mensagens correspondentes à invocação. Esses interceptadores atuam, tanto no cliente como no objeto de aplicação servidor.

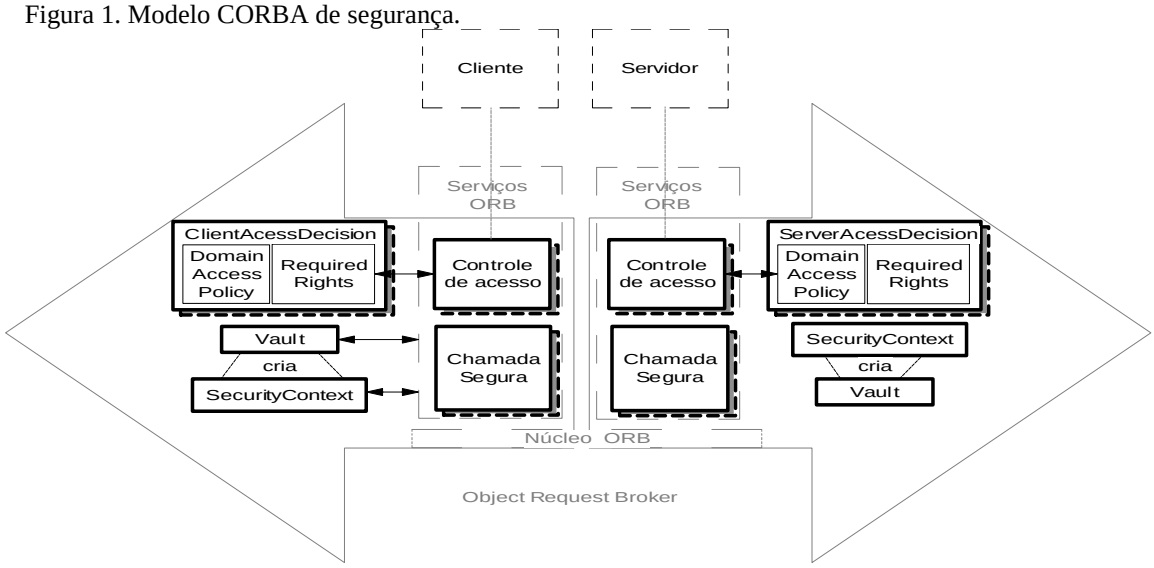


Figura 2. Modelo de Segurança CORBA.

Analisando a Figura 2, observa-se que no interior das setas estão contidos o interceptador de controle de acesso que está representado pelo objeto *Controle de Acesso* e o interceptador de chamada segura que está representado pelo objeto *Chamada Segura*. Os objetos de serviço que implementam os controles de segurança nas especificações CORBA são: o *Principal Authenticator* que corresponde ao serviço de autenticação de

[2] Mecanismos oferecido pelo ORB, de propósito geral, interposto no caminho de invocação, ou resposta, entre um cliente e um objeto alvo, sendo responsáveis pela ativação transparente de controles ou processamentos especiais às quais estariam sujeitas as invocações normais no ambiente CORBA.

principais no CORBA; o *Vault* que estabelece as associações seguras entre os clientes e servidores com os respectivos contextos de segurança; o *SecurityContext* que representa o contexto da associação segura, o *Credential* que representa as credenciais ou direitos do cliente na sessão; o *DomainAccessPolicy* que representa a interface de gerenciamento de políticas de autorização discricionárias e concede a um conjunto de principais direitos específicos para executar operações sobre todos os objetos do domínio [7]; o *RequiredRights* que armazena as informações sobre os direitos (*rights*) necessários para executar cada método de cada interface do sistema; e os objetos *AccessDecision* responsáveis por interagir com os objetos *DomainAccessPolicy* e *RequiredRights* para determinar se uma dada operação sobre um objeto destino específico é permitida. Existem, também, outros objetos de serviço relacionados com a não-repudiação e Auditoria.

O objeto *Chamada Segura* é o responsável em tempo de ligação (*bind time*) pelo estabelecimento da associação segura entre o cliente e o servidor. Em tempo de ligação, esse interceptador de baixo nível ativa o objeto de serviço *Vault* no sentido de criar um objeto *SecurityContext* da associação segura que deve ser estabelecida entre os objetos cliente e servidor. O objeto *SecurityContext* fornece as informações do contexto de segurança estabelecido e é usado pelo objeto *Chamada Segura* na proteção de mensagens para garantir integridade e/ou confidencialidade durante as invocações normais.

3. Tecnologias de segurança

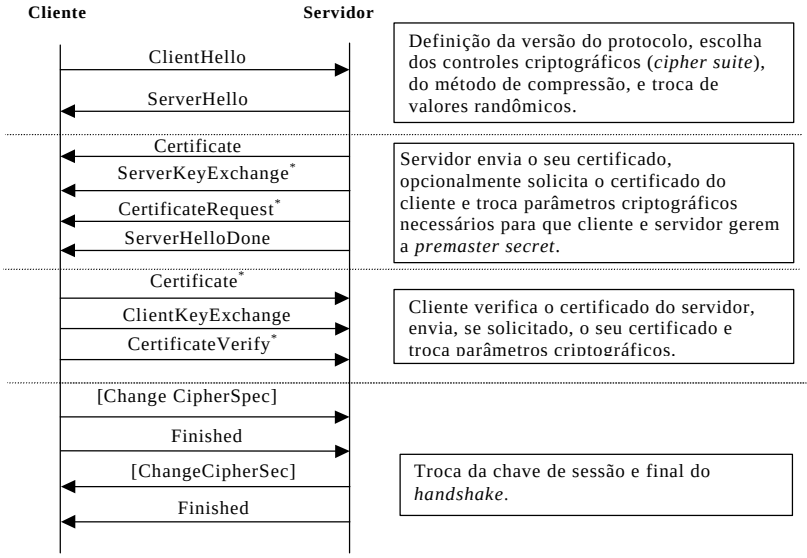
O s objetos de serviço no modelo CORBA de segurança isolam aplicações e o ORB da tecnologia de segurança (Figura 1). Esta última consiste em uma camada subjacente que implementa várias funcionalidades dos objetos de serviço relacionados com a segurança. A tecnologia de segurança inclui serviços de autenticação, serviços de associação segura (distribuição de chaves, certificados, cifragens/decifragens), etc. De acordo com a especificação do *CORBASec*, as tecnologias que podem ser utilizadas para fornecer esses serviços são [15]:

- **SPKM** (*Simple Public-Key GSS-API Mechanism*): suporta políticas baseadas em identidade sem delegação (*Common Secure Interoperability* - CSI nível 0). Essa especificação define protocolos, procedimentos e convenções a serem empregadas em conjunto com a interface GSS-API na utilização de um mecanismo de chave pública simples (*Simple Public-Key*). O uso do SPKM no CORBA requer a extensão SECIOP (*Secure Inter-ORB Protocol*);
- **Kerberos**: suporta políticas baseadas em identidade com delegação irrestrita (CSI nível 1) usando a tecnologia de chave simétrica para a distribuição de chaves. É possível utilizar esse protocolo sem delegação, fornecendo assim o nível 0 do CSI. O uso do Kerberos no CORBA requer a extensão SECIOP (*Secure Inter-ORB Protocol*);
- **CSI-ECMA** (baseado no SESAME e no mecanismo ECMA GSS-API): suporta políticas baseadas em identidade e privilégio, com delegação controlada (CSI nível 2). O protocolo pode ser usado com identidade do principal, mas sem outros privilégios e sem restrições de delegação (CSI nível 1), ou ainda pode ser usado sem delegação (CSI nível 0). O uso do CSI-ECMA no CORBA requer a extensão SECIOP (*Secure Inter-ORB Protocol*);
- **SSL**: fornece apenas a funcionalidade CSI de nível 0, ou seja, políticas baseadas em identidade. Esse nível de funcionalidade é atingido apenas se as características de autenticação opcionais do SSL forem utilizadas. Neste caso, já que o SSL fornece uma camada de transporte segura sobre o TCP/IP, o protocolo SECIOP não é necessário quando o SSL é utilizado.

3.1 O protocolo SSL

O SSL [2] fornece autenticação, confidencialidade e integridade para comunicações através de conexões TCP/IP. A autenticação permite que uma aplicação verifique a identidade da outra aplicação com a qual está se comunicando. Confidencialidade garante que os dados transmitidos entre aplicações não poderão ser entendidos por uma parte intermediária. Integridade possibilita que as aplicações detectem os dados que podem ter sido modificados durante a transmissão. Este protocolo está localizado entre o protocolo da camada de transporte, por exemplo, o TCP e o protocolo da camada de aplicação, e é composto por duas camadas: o protocolo ***SSL Record***, que provê conexões seguras garantindo confidencialidade e integridade das mensagens transportadas, através de criptografia simétrica e de mensagens de integridade (MAC – *message aunthentication code*); e o protocolo ***SSL Handshake***, que permite a autenticação de cliente e servidor, negocia algoritmos criptográficos e gera a chave simétrica usada pela camada *SSL Record*.

O SSL utiliza, para autenticação de cliente e servidor, certificados digitais e transferências com assinaturas digitais. A sessão SSL é estabelecida quando a negociação inicial (*handshake*) entre cliente e servidor é realizada. O fluxo de mensagens durante o *handshake* está representado na Figura 3.



* Indica que a mensagem é opcional, ou depende da situação para ser enviada.

Figura 3. Fluxo de mensagens do *handshake*.

A autenticação no SSL usa o X.509 *certificates* [6] para transferir informações sobre a chave pública de uma aplicação. O SSL v3 usa o *X.509 v3 certificates* para assegurar chaves RSA, e um *X.509 certificates* modificado para assegurar chaves públicas usadas pelo protocolo de distribuição de chaves do Departamento de Defesa dos EUA, *Fortezza/DMS*.

O X.509 *certificates* possui autoridades certificadoras localizadas no mundo todo. O X.509 é um padrão internacional que tem aplicabilidade em diversos campos, tendo forte aceitação internacional.

4. *Framework JaCoWeb Security*

A definição do *framework* de integração do ORB com suporte SSL partiu do estudo e da análise detalhada da estrutura de um ORB convencional. O JacORB [3] foi o ORB em Java escolhido para ser usado no *framework JaCoWeb*. Esse *middleware* corresponde a um conjunto de suportes e ferramentas de desenvolvimento que permitem construir e integrar aplicações orientadas a objetos em sistemas abertos. O JacORB é completamente desenvolvido em Java, trazendo as vantagens desta linguagem, seguindo as especificações da OMG.

Na figura 4, é apresentado o modelo de implementação do ORB/CORBA do projeto JacORB. Em detalhe, é apresentado o fluxo de uma invocação dentro do ORB, a partir do ponto em que os dados da requisição (em *byte stream*) alcançam o *socket* no lado servidor. O módulo *BasicAdapter*, através do objeto *Connection*, trata das funções de baixo nível que manipulam as conexões TCP/IP entre clientes e servidor (os objetos *ServerSocket*). É importante ressaltar que é criada apenas uma conexão para cada par cliente/servidor, não importando quantos objetos no servidor estão sendo utilizados pelo cliente. O objeto *RequestReceptor* faz a decodificação (*unmarshalling*) do conjunto de *byte stream* para re-montar o objeto *Request* no lado servidor. Então, o *Request* é transformado em *ServerRequest* para ser mandado para a fila de requisições (*RequestQueue*), onde é ordenado em seqüência para chegar ao POA (*Portable Object Adaptor*).

Conceitualmente, o POA adapta o conceito da linguagem de programação utilizada para implementar um *servant* ao conceito de objeto CORBA [15]. Um *servant* é uma entidade com uma linguagem de programação particular (por exemplo, um objeto em Java ou uma coleção de funções em C). Um objeto CORBA é uma representação remota de um objeto *servant* diante do ORB – permite, deste modo, que o *servant* possa ser identificado, localizado e endereçado por uma referência de objeto (IOR). As principais funções do POA são geração e interpretação de referência de objeto, invocação de métodos, ativação, desativação de objeto e implementações, mapeamento de referência de objeto para implementação, etc.

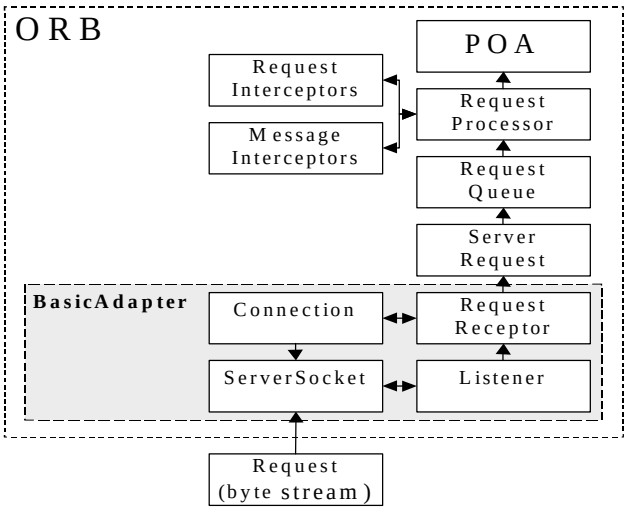


Figura 4. Fluxo de dados no servidor.

Durante a fase de projeto foram discutidas duas formas de integrar a tecnologia SSL ao ORB, ambas de acordo com as recomendações definidas pela especificação CORBASec [15]. Neste trabalho procurou-se definir uma solução que permitisse ao ORB tratar tanto conexões seguras como convencionais, deste modo mantendo, também, as funcionalidades originais do ORB. A primeira alternativa consiste em implementar toda a especificação do SSL como sendo parte do serviço de segurança (Figura 2). Neste caso, o modelo é inteiramente respeitado e os contextos de segurança são criados e controlados pelo serviço de segurança. Ou seja, o interceptador de mensagem tem o controle total do processo de *handshake* (item 3.1). O maior obstáculo desta solução é o alto custo de implementação. A outra solução consiste em utilizar um pacote SSL com código executável disponível e adequá-lo as nossas necessidades. Neste caso, perde-se um pouco da potencialidade do interceptador de mensagem no controle do *handshake*.A segunda alternativa foi escolhida como solução já que a não implementação de toda a especificação do SSL permitiu aos autores dedicar especial atenção aos outros serviços do modelo de segurança CORBA [18].

4.1 ORB+SSL

O *framework* para a integração do ORB/CORBA com suporte SSL consiste em encapsular um pacote SSL já implementado na forma de objeto de serviço, tal como os objetos COSS. Escolheu-se o pacote iSaSiLk [4] por apresentar todas as funcionalidades necessárias para o *framework* proposto. Na figura 5, são apresentados os objetos, com interfaces definidas em IDL, que compõem o pacote *JaCoWeb Security* para a integração ORB+SSL. A classe *Security* encapsula as funcionalidades do pacote iSaSiLk e é instanciada pela aplicação no momento de sua criação. O conjunto de parâmetros definidos na criação do objeto *Security* permite ao iSaSiLk definir o *Contexto SSL* (figura 5). O *Contexto SSL* define como cliente e servidor devem selecionar de forma compatível os controles criptográficos (*Cipher Suite*, figura 3).

Na figura 5, têm-se os objetos *ServerSocket* e *SSLServerSocket*: um para estabelecer conexões normais através do *Listener* e outro para estabelecer conexões seguras através do *SSLListener*. O *SSLServerSocket* é fornecido pelo iSaSiLk e é encapsulado pelo JaCoWeb. O *SSLConnection* tem a função de instanciar o *SSLSocket* para o estabelecimento da conexão segura entre cliente e servidor. Portanto, o estabelecimento da chamada segura é realizado dentro do pacote JaCoWeb e, como resultado, o conjunto de *byte stream* da requisição é passado ao ORB através do *SSLRequestReceptor*.

Convém notar que ambos *Listener* encaminham o conjunto de *byte stream* para o seu próprio objeto receptador de requisição (*RequestReceptor* e *SSLRequestReceptor*, respectivamente) que realiza a conversão para remontar o objeto *Request*. A partir desse momento a requisição pode seguir seu caminho normal até o POA, onde ocorre a transformação dos parâmetros do objeto *Request* em argumentos a serem passados ao *servant*. O interceptador de requisição tem a função de ativar o serviço de segurança para realizar o teste de controle de acesso na invocação de um método [18].

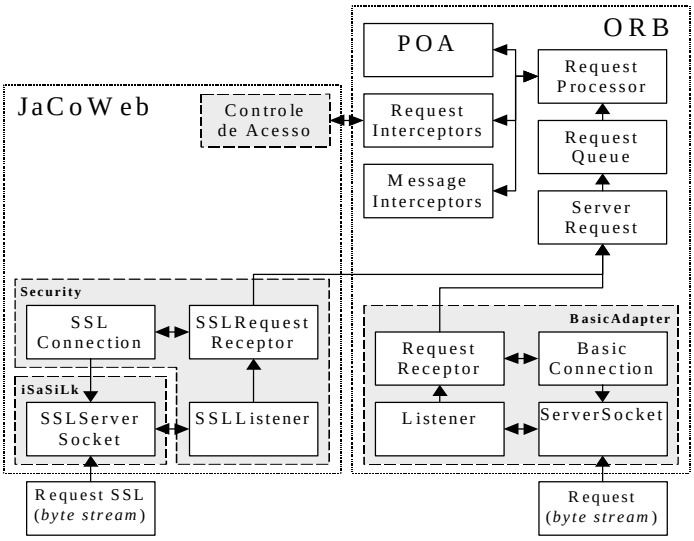


Figura 5. O *framework* ORB+SSL.

O ponto chave para esta integração dar certo é definir como a aplicação pode selecionar o modo de conexão a ser utilizado (segura ou convencional). No modelo CORBA, um objeto pode ser localizado nos sistemas distribuídos através da sua referência de objeto. Na OMG, a referência de objeto é definida numa forma padrão conhecida como IOR (*Interoperable Object Reference*). A IOR [14] é uma seqüência de caracteres que quando convertida para o formato adequado fornece as informações de endereço IP do sítio, da porta, de um ponteiro para o objeto a ser acessado e algumas informações de controle relacionadas à própria IOR.

A especificação CORBASec define uma extensão da IOR de forma que possa agregar informações especificas para o ORB tratar, também, de conexões seguras. A extensão da IOR é feita adicionando um novo tipo de *Tag* (etiqueta) de segurança ao corpo do IIOP *profile* da IOR. Vale ressaltar que, apesar do servidor ter disponíveis duas portas de acesso (uma segura e outra convencional), qualquer tentativa indevida de acesso a um serviço (objeto) com restrições de segurança pela porta convencional é inibida pelo serviço de controle de acesso no interceptador de requisição (Figura 5).

5. Implementação do *JaCoWeb Security*

São discutidos nesta seção, os aspectos relacionados à implementação do ORB+SSL. A implementação adota as especificações COSS/OMG referente ao serviço de segurança (*CORBASec*). Toda a implementação teve como plataforma de desenvolvimento o JacORB beta 13.

5.1 Interfaces IDL do *Framework*

Quando as aplicações estabelecem associações seguras utilizando o *framework*, algumas informações de segurança são inicializadas. O objeto *Security* é o responsável pela inicialização da segurança de acordo com a IDL da Figura 6. Após a inicialização do ORB e do POA (somente para servidores), as aplicações devem completar a inicialização do ORB seguro instanciando um objeto *Security*, de acordo com o seu tipo de aplicação (cliente *stand-alone*, *applet* ou servidor).

A linguagem Java popularizou o conceito de código móvel através de seus *applets* executados a partir de browsers Web. No contexto deste trabalho, as aplicações Web, tais como comércio eletrônico e *internet banking*, são expressas na forma de clientes representados por *applets* Java[3], e de servidores CORBA (objetos remotos) que interagem através de um ORB seguro. Para aplicações Web, as informações de segurança são definidas através do primeiro método da interface IDL, apresentada na Figura 6 (`initAsClient (orb,applet)`). Este método instancia um objeto Security e invoca o método `initAsClient (orb)`.

[3] *Applets* clientes estão sujeitos às restrições de segurança impostas pelo *browsers* onde são executados. Estas restrições impedem que esses códigos possam ter acesso ao disco local ou façam conexões com sítios (*hosts*) diferentes dos que eles foram carregados. De alguma maneira isto impõe que os objetos servidores da aplicação estejam na mesma máquina do servidor Web a partir do qual o *applet* foi carregado. Algumas técnicas são oferecidas para permitir que os applets se libertem dessas restrições e possam se comunicar com objetos em um sistema distribuído sem limitação de localização. Neste trabalho, adotou-se a solução de *applets* assinados da plataforma Java™ [17]. Na versão Java 1.2, a segurança é controlada pela política de segurança aplicada em tempo de execução. Portanto, pode-se configurar o arquivo de política para que privilégios sejam concedidos aos *applets*, livrando-os das restrições impostas.

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

Figura 6. Interface IDL do pacote JaCoWeb

Para aplicações *stand-alone*, o contexto de segurança pode ser definido através do método `initAsClient (orb)`, que deve: a partir do certificado do cliente definir o principal; criar a credencial SSL estática do cliente (com base no certificado deste cliente); criar e definir os objetos de política utilizados no estabelecimento da associação segura; definir o contexto SSL do cliente, especificando os *cipher suites*, o objeto *TrustDecider* (responsável pela autenticação do servidor) e o certificado do cliente, e finalmente definir estas configurações de segurança (objeto *Security*) na classe jacorb.orb.Environment.

A figura 7 apresenta a *Tag* com definições para conexões SSL. Basicamente, a estrutura de dados associada com o componente SSL encontra-se definida no módulo SSLIOP [15]. O aumento do tamanho da IOR não traz nenhum problema de compatibilidade adicional, uma vez que a estrutura da IOR é bastante flexível e o próprio serviço de nomes já suporta IORs de tamanhos variados.

```
module SSLIOP {
  // Security mechanism SSL
     const IOP:: ComponentID       TAG_SSL_SEC_TRANS = 20
     struct SSL {
       Security::AssociationOptions       target_supports;
       Security::AssociationOptions       target_requires;
       unsigned short                             port;
 };};
```

Onde:

- **AssociationOptions**: são as opções de associação que podem ser *NoProtection*, *Integrity*, *Confidentiality*, *DetectReplay*, *DetectMisordering*, *EstablishTrustInClient*, *EstabilishTrustInTarget*;
- **target_supports**: fornece as funcionalidades suportadas pelo objeto destino;
- **target_requires**: define o mínimo de funcionalidades esperadas pelo objeto destino;
- **port**: número da porta TCP/IP que será usada nas conexões.

Figura 7. Estrutura do *Tag* com definições para conexões SSL

Para estabelecer uma associação segura utilizando o SSL, o servidor da aplicação deve registrar-se no Serviço de Nomes com uma IOR que contenha a *Tag* SSL. Para isto, modificações no método ORB.createIOR() do JacORB foram necessárias. E, como os clientes necessitam conhecer a porta SSL e os requisitos necessários para estabelecer a conexão segura, a classe ParsedIOR, responsável pela decodificação da IOR, também foi alterada.

6. Avaliação do *JaCoWeb Security*

O esquema de autorização JaCoWeb, usando políticas de controle de acesso discricionárias, foi avaliado segundo o Critério Comum para avaliação de segurança [19]. O Critério Comum (***C**ommon **C**riteria – CC*) para avaliação da segurança é o resultado do esforço de várias organizações internacionais para desenvolver um único critério de avaliação da segurança para sistemas e produtos de tecnologias de informação em sistemas distribuídos. O CC [20] define como expressar os requisitos de segurança de um sistema ou produto, define categorias distintas de requisitos funcionais (*functional requirements*) e requisitos de garantia (*assurance requirements*). Os *requisitos funcionais* definem o comportamento de segurança desejado. Os *requisitos de*

garantia são o fundamento para se conquistar a confiança de que as medidas de segurança solicitadas estão efetiva e corretamente implementadas. A confiança na segurança de uma tecnologia de informação pode ser obtida através das ações realizadas durante o processo de desenvolvimento, avaliação e operação.

As ameaças (***T****hreats*) que o protótipo espera impedir são designadas da seguinte forma: *T.ACCESS*, onde um usuário obtém acesso a informações ou recursos do sistema sem ter a devida permissão, *T.CAPTURE*, onde um invasor pode modificar ou obter informações pela escuta da linha de transmissão, *T.INTEGRITY*, onde a integridade das informações transmitidas pode ser comprometida devido a erros do usuário ou de transmissão, *T.SECRET*, onde um usuário do protótipo pode obter informações do sistema para as quais não tem permissão de maneira intencional ou acidental e *T.IMPERSON*, onde um invasor pode obter acesso a informações ou recursos por se fazer passar por um usuário autorizado do protótipo. Alguns tipos de ameaças não podem ser contidas no protótipo construído: o abuso por parte dos usuários autorizados de forma que ocorra consumo excessivo dos recursos e, consequentemente, negação de serviço a outros usuários autorizados, e, não há como responsabilizar um usuário sobre os atos cometidos pois serviços de auditoria não estão presentes.

A avaliação considerou o *JaCoWeb Security* nível EAL3. Isso quer dizer que o protótipo foi metodicamente testado e verificado, ou seja, o protótipo é aplicável a circunstâncias onde se necessita um nível moderado de segurança. O foco principal deste artigo foi a integração da tecnologia SSL ao ORB, e avaliar o seu desempenho. A análise do desempenho completa a análise qualitativa feita em [19].

6.1 Desempenho do *JaCoWeb Security*

Neste item são apresentadas as análises de desempenho do *JaCoWeb Security*, realizadas no laboratório já mencionado, com o sentido de avaliar o desempenho da implementação. As medidas foram realizadas com um servidor e um cliente. O objetivo dessas medidas é avaliar o custo adicional ao integrar mecanismos de segurança (SSL) ao ORB. A fim de possibilitar essa análise foram comparadas as medidas realizadas em um ORB convencional (JacORB) com o JaCoWeb Security.

A primeira medida realizada foi a da latência. No *JaCoWeb*, quando o cliente faz a primeira requisição no servidor, é executado um conjunto de procedimentos para o estabelecimento do contexto de segurança SSL (figura 3). O algoritmo de chave pública, usado para autenticação mútua e distribuição de chaves, foi o RSA de 1024 bits. Para cifragem dos dados utilizou-se o algoritmo simétrico de cifragem em bloco 3DES_EDE e a função *hash* de segurança SHA para a computação do MAC.

Standalone	Tempo (ms)
JaCoWeb (primeira chamada)	1600
JacORB (primeira chamada)	4
JaCoWeb (média das 1000 chamadas subsequentes)	8
JacORB (média das 1000 chamadas subsequentes)	4

Rede Local (Ethernet)	Tempo(ms)
JaCoWeb (primeira chamada)	1700
JacORB (primeira chamada)	5
JaCoWeb (média das 1000 chamadas subsequentes)	10
JacORB (média das 1000 chamadas subsequentes)	5

Figura 8. Latência do JaCoWeb e JacORB.

A figura 8 apresenta duas tabelas com os tempos, em milisegundos, da média de 1000 *pings*[4] entre cliente e servidor. A primeira apresenta as medidas realizadas em um computador Pentium II 300 Mhz com cliente e servidor executando na mesma máquina (*standalone*). Esta medida permite avaliar o custo de processamento adicional relativo às computações do código SSL.

A segunda tabela apresenta as medidas realizadas, de forma distribuída, em uma rede local (com 10 Mbps *Ethernet*). Os tempos são medidos com um cliente em um computador Pentium II 300 Mhz e um servidor em um computador Pentium II 400 Mhz.

Nesta situação, é considerado, além do custo de processamento do SSL, o custo relativo ao tempo gasto nas interações sobre a rede para o estabelecimento do contexto SSL. É interessante observar que a primeira chamada no JaCoWeb é relativamente alta (aproximadamente 1600 ms de processamento mais 100 ms para estabelecimento do contexto SSL, totalizando 1700 ms). Isto é o resultado do processamento computacional para a cifragem e decifragem com chaves públicas, que é requerida para inicializar a primeira conexão do SSL. No

[4] Ping (*Packet INternet Groper*) – é um mecanismo que, entre outras funções, pode ser usado para testar a qualidade da ligação ou para medir o tempo de resposta (reconhecimento) de uma chamada.

entanto, para as chamadas (os *pings*) subseqüentes, o tempo de aproximadamente 10 ms é considerado satisfatório.

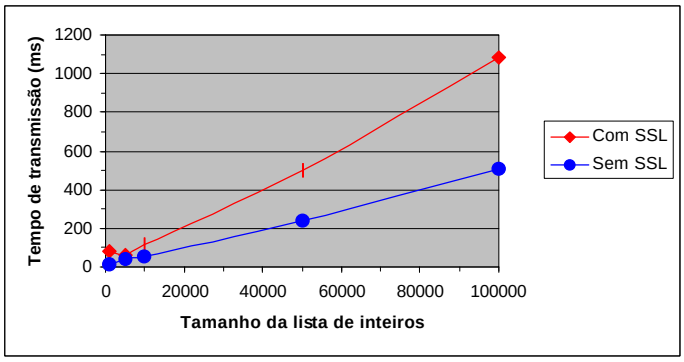


Figura 9. Tempo de envio de pacotes do JaCoWeb e JacORB.

A outra medida realizada foi a do tempo de envio dos pacotes (figura 9), com diferentes tamanhos, entre um cliente e um servidor nas mesmas configurações de hardware. Os tamanhos dos pacotes foram 1, 5, 10, 50 e 100 Kbytes. Pode-se observar que os tempos medidos para o *JaCoWeb* usando o SSL são praticamente o dobro do obtido por um ORB convencional (*JacORB*). Estes resultados refletem o custo do suporte de segurança para cifragem das mensagens transmitidas sobre a rede.

7. Considerações gerais

Neste item avaliamos o modelo de integração ORB+SSL, de acordo com os requisitos levantados na especificação CORBASec [15] e nas considerações dessa especificação, apresentadas em [8]. Os requisitos, nos quais está baseada esta avaliação, são: transparência, simplicidade, reusabilidade, escalabilidade, flexibilidade e interoperabilidade.

A transparência oculta as funcionalidades (mecanismos) do sistema de segurança do usuário e do programador da aplicação, dando a impressão de que as invocações são atendidas localmente. O aspecto da simplicidade indica que o sistema de segurança deve ser fácil de entender, usar e gerenciar. O *JaCoWeb Security* tenta atender a esses dois requisitos da melhor maneira possível, definindo um objeto *Security* (item 5.1), responsável pela inicialização e ativação de todos os mecanismos de segurança. A partir daí, as invocações são realizadas tal como num ORB convencional. No aspecto da facilidade de gerenciamento das configurações e políticas de segurança, o *JaCoWeb* é implementado de forma que as configurações de sistema sejam definidas estaticamente, em tempo de compilação, na classe *Enviroment*. No entanto, as políticas de segurança, para o controle de acesso às aplicações, podem ser definidas dinamicamente em tempo de execução através de uma interface de gerenciamento.

A reusabilidade é uma consideração importante, já que diminui o tempo de desenvolvimento e manutenção dos programas, tornando-os mais robustos e confiáveis. A infra-estrutura de segurança do *JaCoWeb*, por ser baseada no modelo de objetos de serviço, pode ser facilmente adaptada para diferentes aplicações. Todos os serviços são descritos em IDL, padrão OMG, o que contribui não só no aspecto do reuso de código como na portabilidade.

O aspecto da escalabilidade diz respeito à possibilidade do sistema de segurança se adequar tanto para sistemas pequenos ou amplos quanto ao número de usuários e operações, isto é, tenha suporte para domínios de políticas, grupos, *roles* etc. A implementação desenvolvida usa conceitos de grupo e *roles* dentro de um domínio de política único, de maneira simplificada.

Em [8] é discutido a relação entre os requisitos de flexibilidade e interoperabilidade. A máxima flexibilidade é alcançada quando o número de interfaces e protocolos padronizados é o menor possível, permitindo maior liberdade aos desenvolvedores e usuários para estender os mecanismos de segurança de acordo com suas necessidades. De modo contrário, a máxima interoperabilidade é alcançada quando se tem um grande número de interfaces e protocolos padronizados. Isto assegura que ORBs de diferentes companhias ofereçam as mesmas funcionalidades, motivando que aplicações, em ORBs distintos, possam interagir.

É claro que existe a dificuldade em atender completamente a cada um desses requisitos sem afetar o outro. O *JaCoWeb Security* se utiliza das opções que a especificação CORBASec oferece para atender, de forma equilibrada, esses requisitos. Novamente, o uso de interfaces IDL e o modelo de objeto de serviço (COSS) na implementação do *JaCoweb Security* nos permitiram atender melhor o aspecto da interoperabilidade. No entanto, como as especificações CORBASec padronizam uma grande parte de todas as abstrações de segurança

(mecanismos, propriedades, políticas, estrutura de dados etc.), o aspecto da flexibilidade é bastante afetado. Contudo, isto não impede que extensões proprietárias sobre o modelo sejam implementadas na forma de objeto de serviço – sem alterar as características do ORB em si.

Em relação a comunicação, no estabelecimento do contexto de segurança através das tecnologias de segurança subjacentes (Figura 1), o *JaCoWeb Security* adota o SSL juntamente com os protocolos de adaptação SSLIOP (*Secure Socket Layer Inter-ORB Protocol*) e SIOR (*Security-enhanced Inter-ORB Reference*), padrões da OMG. Isto significa que a interoperabilidade com outros ORBs só é alcançada quando ambos utilizam os mesmos protocolos (SSLIOP) e tenham políticas de segurança "consistentes" [8]. Isto exclui ORBs que utilizam o protocolo de adaptação SECIOP (para SPKM, Kerberos e CSI-ECMA – no item 3) e DCEIOP (para DCE). No entanto, o *framework JaCoWeb Security* foi modelado de forma a permitir uma fácil adaptação às outras tecnologias de segurança, desde que tenha uma API disponível.

Em termos de experiências acadêmicas de implementação em Java do modelo de segurança CORBA, pode ser encontrada em *Nephilim: Java Implementation of CORBA Security Services* [9]. Este trabalho utiliza **CSI-ECMA** (baseado no SESAME e ECMA GSS-API) como tecnologia de segurança e suportando CSI nível 2. Outro projeto, o *JacORBoverSSL*, desenvolvido por Andre Benvenuti, também é uma implementação em Java que, assim como este trabalho, visa integrar um ORB ao SSL. Entretanto, a implementação *JacORBoverSSL* foi realizada no núcleo do ORB, ou seja, a tecnologia de segurança introduzida não está de acordo com os níveis do modelo de segurança CORBA.

Além destes protótipos, existem alguns produtos comerciais que integram o ORB a uma tecnologia de segurança. Podem ser destacados Jbroker [10], ORBAsecSL2 [1], ORBIXSecurity [5] e SecureBroker SSL [16]. O **Jbroker** fornece o suporte para o uso do IIOP sobre o SSL, entretanto, apesar de utilizar o protocolo SSLIOP, a manipulação do protocolo SSL é toda realizada pelo núcleo do ORB. Analisando o **OrbixSecurity** [18], verifica-se que a implementação deste ORB seguro garante a segurança da comunicação através do uso do SSL. Constatou-se que esta implementação segue as interfaces definidas no *CORBAsec*, sem implementar interceptadores padrão, por outro lado utiliza os *filters* e *transformers* como mecanismos de desvio. De acordo com [1] e [16], o **ORBAsecSL2** e o **SecureBroker** cumprem perfeitamente o modelo CORBAsec. Ambos usam o Kerberos e o SSL como tecnologia de segurança. O SecureBroker ainda está em fase de desenvolvimento, somente a integração com o SSL está disponível (SecureBrokerSSL) e ainda em fase de testes.

8. Conclusão

Neste trabalho são discutidas soluções para a adição de mecanismos de segurança no ORB. Com isso, foi proposto um *framework* geral para a integração de tecnologias de segurança ao ORB que não comprometem o aspecto da interoperabilidade do ORB como um todo. Este *framework* atua no sentido de adaptar uma tecnologia de segurança, tal como uma API que chama os mecanismo de segurança do serviço de segurança subjacente.

A plataforma CORBA pode ser usada para integrar aplicações e sistemas, fornecendo a flexibilidade necessária para os ambientes de negócios que mudam dinamicamente, como os encontrados na Web. O uso do SSL como tecnologia de segurança subjacente garante às aplicações Web críticas as características desejáveis de integridade, confidencialidade e autenticidade das informações e recursos. Para estas aplicações ainda é preciso garantir que os recursos não serão usados por pessoas não autorizadas ou de formas não autorizadas. Neste caso, mecanismos de autorização, que também compõem o modelo CORBASec, podem ser empregados. Em [18], tem-se a descrição do esquema de autorização proposto pelo projeto *JaCoWeb Security*.

Além disso, neste trabalho também são desenvolvidas as implementações do *framework* JaCoWeb sobre as quais são realizadas análises de desempenho. Estas avaliações apontaram os custos necessários de desempenho para garantir conexões seguras sobre o ORB, usando a tecnologia SSL. Estas implementações podem ser obtidas na WEB no seguinte endereço (www.lcmi.ufsc.br/jacoweb).

Bibliografia

[1] Adiron, LLC**., "ORBaSecSL2 – User Guide**", version 2.0, April 1999. http://www.adiron.com/download.html

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

[3] G. Brose, **"JacORB: Implementation and Design of a Java ORB"**, Procs. of DAIS'97, IFIP WG 6.1 International Working Conference on Distributed Aplications and Interoperable Systems, Cottbus, Germany, Chapman & Hall, Sep. 1997.

[4] IAIK-Java Group, **"iSaSiLk 2.5 User Manual"**, Institute for Applied Information Processing and Communications, Graz, University of Technology. http://jcewww.iaik.at/iSaSiLk/document.htm, Nov. 1999.

[5] Iona Technologies, "**Orbix Security 3.0 - White Paper**", September 1999. http://www.iona.com/support/whitepapers/download.html

[6] ITU-T, **"Information Technology – The Open Systems Interconnection – The Directory: Autentication Framework"**, ITU-T Recommendation X509, Nov. 1993.

[7] 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.

[8] U. Lang and R. Schreiner, "**Flexibility and Interoperability in CORBA Security"**, eletronic notes in Theoretical Computer Science Vol. 32, Elsevier Science B.V., 2000.

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

[10] ObjectEra, "**Jbroker 2.1 Tutorial**".http://www.objectera.com/jbroker/jbroker.html

[13] Object Management Group, **"The Common Object Request Broker 2.0/IIOP Specification"**, Revision 2.0, OMG Document 96-08-04, 1996.

[14] Object Management Group, **"CORBAservices: Common Object Services Specification"**, OMG Document March, 1997. http://www.omg.org.

[15] Object Management Group, **"Security Service: v1.2 Final"**, OMG Document Number 98-01-02, in CORBAservices, Nov. 1998 . http://www.omg.org .

[16] Promia, "SecureBroker". http://www.promia.com/products/securebroker.html

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

[18] C. M. Westphall and J. S. Fraga, **"Authorization Schemes for Large-Scale Systems based on Java, CORBA and Web Security Models"**, The IEEE International Conference on Networks – ICON'99, pp. 327-334, Sep. 1999, Brisbane-Queensland, Australia.

[19] C. M. Westphall and J. S. Fraga, **"Policap – Um Serviço de Política para o Modelo CORBA de Segurança"**, XVIII Simpósio Brasileiro de Redes de Computadores (SBRC'2000), Maio 2000, Belo Horizonte, Brasil.

[20] CC Project Sponsoring Organisations, "Common Criteria for Information Technology Security Evaluation," *In Part2: Security Funcional Requirements*, ISO/IEC 15408-2, December 1999.