Implementação e Avaliação de Desempenho *JaCoWeb Security*

Luciana Moreira Sá de Souza , Ricardo Sangoi Padilha, Joni da Silva Fraga
Laboratório de Controle e Microinformática (LCMI) – DAS – CTC – UFSC
Campus Universitário – Caixa Postal 476 – Trindade – CEP 88040-900 – Florianópolis – SC – Brasil
e-mail: {luciana, padilha, fraga}@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. 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*

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*

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 sociedade. 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.

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 de comunicação, dentre outras que possam existir [1]. Vulnerabilidades podem ocorrer entre cada uma dessas camadas, pois a interação entre as mesmas é bastante complexa. A complexidade é ainda maior em sistemas onde a garantia da segurança é amplamente distribuída. O paradigma dos objetos distribuídos em sistemas abertos implementado pelo uso de novas ferramentas como Java, CORBA e Web, introduziram novos problemas de segurança implicando a necessidade de novos modelos e conceitos.

O padrão CORBA (*Common Object Request Broker Architecture*) [1] é adotado neste trabalho no sentido de atender aos requisitos de sistemas abertos. Esta plataforma pode ter sua funcionalidade extendida através de especificações COSS (*Common Object Service Specification*) [2]. Entre estas, temos o serviço de segurança que define um conjunto de mecanismos para autenticação, autorização, auditoria, segurança da comunicação e não-repudiação. A autenticação permite a verificação da identidade da outra parte. 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.

No entanto, acredita-se que estas especificações não apresentam uma visão clara, em termos de conceitos e necessidades, sobre a integração do ORB (*Object Request Broker*) com as tecnologias de segurança sem comprometer o aspecto da interoperabilidade. 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, em sua última revisão do serviço de segurança [3], como um dos protocolos a serem utilizados por aplicações que implementam a segurança no CORBA.

É proposto neste artigo um *framework JaCoWeb Security* que integra, baseado nas especificações de segurança do CORBA [3], 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. Além disso, são discutidos alguns aspectos da implementação e medidas de desempenho no sentido de avaliar o custo desta integração.

Este artigo está dividido em 7 seções: na seção 2 o serviço de segurança da OMG é descrito; 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; e finalmente, na seção 7, as conclusões deste 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 heterogêneos através de um ORB. O mesmo, num sentido mais genérico, é um canal de comunicação para objetos distribuídos. No CORBA, a interoperabilidade entre objetos é conseguida a partir das especificações das interfaces em IDL (*Interface Definition Language*). O padrão CORBA oferece ainda um conjunto de objetos de serviço (*COSS*) [2] que facilitam o desenvolvimento de aplicações. Esses serviços são padronizados pela OMG dentro da perspectiva da arquitetura OMA (*Object Management Architecture*). Estes serviços são representados por uma coleção de interfaces, a nível de sistema, que oferecem funções básicas para implementação e uso. As especificações COSS podem ser entendidas como extensões ou complementos das funcionalidades do ORB.

2.1 Descrição do serviço de segurança CORBA (CORBA Security)

O CORBA *Security Service* [3] é a especificação da OMG que define o modelo de segurança para objetos distribuídos. Este modelo estabelece vários procedimentos envolvendo a autenticação, a verificação da autorização na invocação de um método remoto, a segurança da comunicação, os aspectos relacionados com esquemas de delegação de direitos, a não-repudiação, a auditoria e a administração da segurança. Neste modelo, os objetos e componentes se relacionam em quatro níveis (figura 1): objetos de aplicação (cliente e servidor); objetos de serviço, serviços ORB e o núcleo do ORB (o *middleware*); componentes de tecnologia de segurança (serviços de segurança subjacentes) e componentes de proteção básica, fornecidos por uma combinação de *hardware* e sistemas operacionais locais. Também são definidos dois tipos de interceptadores1 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.

Os objetos de serviço que implementam os controles de segurança nas especificações CORBA são (figura 2): o *Principal Authenticator* é responsável pelo serviço de autenticação de principais no CORBA, o *Security Context* representa o contexto da associação segura, o *Vault* estabelece as associações seguras entre os clientes e servidores, o *Credential* representa as credenciais ou direitos do cliente na sessão, o *Domain Access Policy* representa a interface de gerenciamento de políticas de autorização discricionárias [10], o *Required Rights* armazena as informações sobre os direitos

[1] Mecanismos oferecido pelo ORB, de propósito geral, interposto no caminho de invocação entre um cliente e um objeto alvo, sendo responsáveis pela ativação transparente de controles ou processamentos especiais.

necessários para executar cada método de cada interface do sistema e os objetos *AccessDecision* interagem com os objetos *Domain Access Policy* e *Required Rights* para permitir o acesso ao objeto de destino. Existem, também, outros objetos de serviço relacionados com a não-repudiação e auditoria. Os interceptadores de controle de acesso e de chamada segura estão representados respectivamente pelos objetos *Controle de Acesso* e *Chamada Segura*.

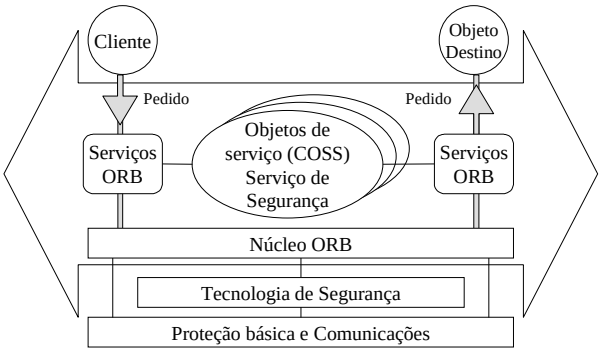

Figura 1: Modelo CORBA de segurança.

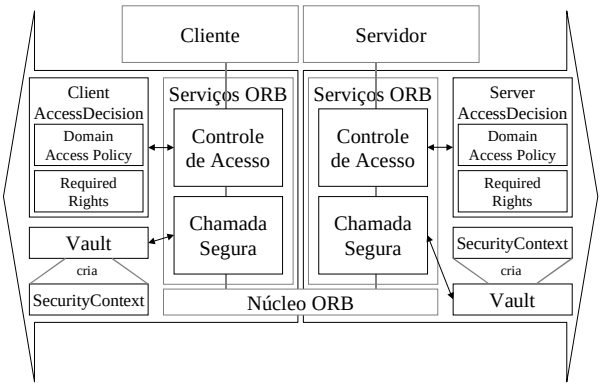

Figura 2: Objetos de serviço de segurança.

Em tempo de ligação entre cliente e servidor, o objeto *Chamada Segura* estabelece a associação segura, ativando o objeto *Vault* no sentido de criar um objeto *Security Context*, que recebe as informações do contexto de segurança e é usado pelo objeto *Chamada Segura* na proteção de mensagens durante as invocações normais.

3. Tecnologias de segurança

Os objetos de serviço no modelo de segurança CORBA isolam as 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 *CORBA Security*, as tecnologias que podem ser utilizadas para fornecer esses serviços são [3]: SPKM, Kerberos, CSI-ECMA e SSL.

3.1 O protocolo SSL

O SSL [5] fornece autenticação, confidencialidade e integridade para comunicações sobre TCP/IP. Este está localizado entre o protocolo da camada de transporte (TCP) e o protocolo da camada de aplicação. É composto por duas camadas: o *SSL Record*, que provê conexões seguras garantindo confidencialidade e integridade através de criptografia e o *SSL Handshake*, que permite a autenticação de cliente e servidor, negocia algoritmos criptográficos e gera a chave usada pela camada *SSL Record*. O SSL utiliza certificados digitais e transferências com assinaturas digitais para a autenticação de cliente e servidor. A sessão SSL é estabelecida quando a negociação inicial (*handshake*) entre cliente e servidor é realizada.

4. Framework JaCoWeb Security

Em um ORB convencional, a invocação de um método em um objeto servidor remoto inicia um conjunto de procedimentos para o preparo da transmissão desta requisição até o servidor. Uma requisição do cliente é transformada em um objeto *Request*, que contém um conjunto de atributos que define o objeto de destino, a operação requisitada, os argumento da invocação, a resposta, etc. Uma vez criado pelo ORB, este objeto passa pelo processo de conversão (*marshalling*) sob forma de *byte stream* para que possa ser enviado pela camada de transporte. No padrão CORBA as mensagens são formatadas de acordo com o GIOP (*General Inter-ORB Protocol*) e transmitidas via IIOP (*Internet Inter-ORB Protocol*), usando o TCP/IP como meio de transporte (GIOP+TCP/IP = IIOP).

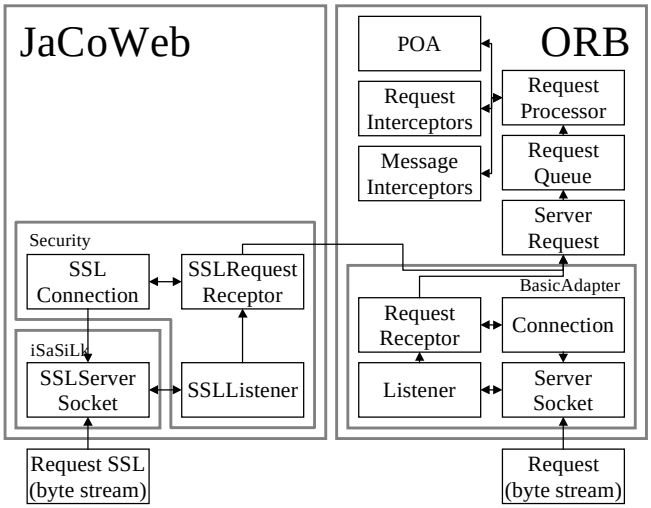

Figura 3: O *framework* ORB+SSL.

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. Escolheu-se o pacote iSaSiLk [8] por apresentar todas as funcionalidades necessárias para o *framework* proposto. Na figura 3, são apresentados os objetos 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 na sua inicialização. Esta classe e os objetos que ela engloba são responsáveis pela implementação dos aspectos de segurança descritos no *framework.*

De uma maneira geral, o pacote de integração tentou estabelecer uma infra-estrutura que reflete a própria estrutura do ORB. Conforme a figura 3, os objetos *ServerSocket* e *SSLServerSocket* processam suas conexões através do *Listener* e *SSLListener*, respectivamente. O *SSLServerSocket* é fornecido pelo iSaSiLk e é encapsulado pelo *Security*. O *SSLConnection* tem a função de receber e manter as informações sobre a conexão segura entre cliente e servidor. O *SSLRequestReceptor* é responsável pela reconstituição do objeto *Request* e sua conversão em *ServerRequest*. Portanto, o estabelecimento da chamada segura é realizado dentro do pacote JaCoWeb e, ao fim do processamento de segurança de baixo nível, os dados da requisição, sob forma de *ServerRequest*, retornam ao seu fluxo normal, dentro do ORB até o POA, onde ocorre a do *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 [9].

No modelo CORBA um objeto pode ser localizado nos sistemas distribuídos através da sua referência de objeto. Esta referência é definida numa forma padrão conhecida como IOR (*Interoperable Object Reference*) [2]. A IOR fornece as informações da máquina como seu endereço IP, a porta, o ponteiro para o objeto de destino. A especificação CORBA *Security* define uma extensão da IOR de forma que possa agregar informações especificas para o ORB processar conexões seguras. Esta extensão é feita adicionando um novo tipo de *tag* (etiqueta) de segurança.

Vale ressaltar que, apesar do servidor ter disponível duas portas de acesso (uma segura e outra convencional), qualquer tentativa indevida de acesso a um serviço (ou objeto) com restrições de segurança pela porta convencional é inibida pelo serviço de controle de acesso no interceptador de requisição (figura 3).

5. Implementação do *JaCoWeb Security*

São discutidos nesta seção os aspectos relacionados à implementação do ORB+SSL. Usamos a implementação do ORB/CORBA JacORB 1.0 beta 13 [7] como referência. O JacORB é inteiramente desenvolvido em Java seguindo as especificações da OMG.

5.1 Interfaces IDL do Framework

Para estabelecer associações seguras usando o *framework JaCoWeb Security*, algumas informações de segurança devem ser inicializadas. O objeto *Security* é responsável pela iniciação da infra-estrutura de segurança conforme definido em sua interface IDL (figura 4). Após a iniciação do ORB, as aplicações devem completar a iniciação do ORB seguro instanciando um objeto *Security*, de acordo com o seu tipo de aplicação (cliente *stand-alone, applet* ou servidor).

Para aplicações Web (*applet*), as informações de segurança são definidas através do método *initAsClient(orb, applet)*. Para aplicações *stand-alone*, as informações de segurança são definidas através do método *initAsClient (orb)*, que a partir do certificado do cliente: define o principal e cria a

```
// $Id: Security.idl, v1.0 1999-12-08
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 4. Interface IDL

credencial SSL estática, cria e define os objetos de política utilizados no estabelecimento da associação segura; define o contexto SSL do cliente especificando os *cipher suites*, instancia o objeto *ServerTrustDecider* (responsável pela autenticação) e instancia um objeto *SSLVault*.

Para o servidor é necessário inicializar o POA após o ORB. Então inicializa-se a infra-estrutura de segurança instanciando o *Security* pelo método *initAsServer(orb, poa)*. A estrutura de dados associado com o componente SSL da IOR encontra-se definida no módulo SSLIOP da especificação CORBA *Security* [3]. O aumento do tamanho da IOR não traz nenhum problema de compatibilidade, uma vez que sua estrutura é bastante flexível e o próprio serviço de nomes já suporta IORs de tamanhos variados.

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 sistema de criação de IOR do JacORB foram necessárias. Da mesma maneira foi alterada a parte responsável pela decodificação da IOR no lado do cliente.

5.2 Implementação do cliente Web (*applet*)

Applets clientes estão sujeitos às restrições de segurança dos navegadores onde são executados (*sandbox*), entre estas restrições estão a impossibilidade de acesso ao disco local e a restrição do estabelecimento de conexões apenas com seu servidor Web. Isto impõe que os objetos servidores da aplicação estejam na mesma máquina a partir do qual o *applet* foi carregado, eliminando a característica de sistemas distribuídos.

A técnica utilizada neste trabalho para permitir que o *applet* saia de seu *sandbox* foi a assinatura de *applets* fazendo uso da plataforma Java [11]. A assinatura de *applets* consiste em o cliente estar ciente que o *applet* tentará romper suas restrições de segurança e dar a ele as permissões necessárias. Na versão JDK 1.2 o cliente cria um arquivo de política de segurança para este propósito.

5.3 Desempenho do *JaCoWeb Security*

Neste item são apresentadas as análises de desempenho do *JaCoWeb Security* com o sentido de avaliar a implementação. As avaliações qualitativas do *JaCoWeb Security* como um todo são apresentadas em [13]. 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 o *handshake* SSL (item 3.1). 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.

Stand-alone	**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 5. Latência do JaCoWeb e JacORB.

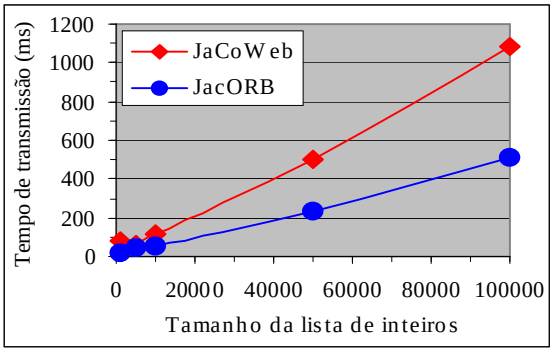

Figura 6. Tempo de envio de listas de inteiros.

A figura 5 apresenta duas tabelas com os tempos, em milisegundos, da média de 1000 *pings* 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 (*stand-alone*). Esta medida avalia 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 (*Ethernet* 10 Mbps). 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. 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 entanto, para as chamadas (os *pings*) subsequentes, o tempo de aproximadamente 10 ms é considerado satisfatório.

A outra medida realizada foi o tempo de envio de pacotes (figura 6) com diferentes tamanhos entre um cliente e um servidor nas mesmas configurações de hardware. O tamanho 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 pelo JacORB. Estes resultados refletem o custo do suporte de segurança para cifragem das mensagens.

6. Considerações

Em termos de experiências acadêmicas relacionadas a este trabalho tem-se o projeto *JacORBoverSSL*, desenvolvido por Andre Benvenuti, que 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, ORBAsecSL2, ORBIXSecurity e SecureBroker SSL [9]. Somente o ORBAsecSL2 e o SecureBroker seguem as especificações do padrão CORBAsec, porém não utilizam a abordagem de objetos de serviço adotada no projeto *JaCoWeb Security*.

Neste item avaliamos o modelo de integração ORB+SSL de acordo com os requisitos levantados na especificação CORBASec [3] e nas considerações, dessa especificação, apresentadas em [14]. 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 do modelo. 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 essas 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 [14] 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, implicando na possibilidade de aplicações, em ORBs distintos, possam interagir.

7. Conclusão

Neste trabalho foram 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 compromete o aspecto da interoperabilidade. Este *framework* pode muito bem ser adotado para integrar outras tecnologias de segurança diferente do SSL.

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 características desejáveis de integridade, confidencialidade e autenticidade das informações e recursos. Também podem ser empregados mecanismos de autorização que compõem o modelo CORBA *Security*. Em [9], tem-se a descrição do esquema de autorização proposto pelo projeto *JaCoWeb Security*.

Além disso, neste trabalho é detalhada a implementação do *framework JaCoWeb*, sobre as quais são realizadas análises de desempenho. Estas avaliações apontaram os custos de desempenho para garantir conexões seguras sobre o ORB usando a tecnologia SSL. Esta implementação pode ser obtida no endereço http://www.lcmi.ufsc.br/jacoweb.

Bibliografia

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

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

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

[4] D. Schmidt and S. Vinoski, *Object Interconnections*, SIGS C++ Report Magazine, Nov/Dec 1997.

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

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

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

[8] 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.

[9] 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.

[10] 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.

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

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

[13] C. M. Westphall and J. S. Fraga, *Policap – Um Serviço de Política para o Modelo CORBA de Segurança*, Submetido ao SBRC2000.

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