JaCoWe – Um Esquema de Autorização para Redes de Larga Escala integrando os Modelos de Segurança Java, CORBA e Web

Carla Merkle Westphall, Joni da Silva Fraga e Lau Cheuk Lung
Laboratório de Controle e Microinformática - LCMI-DAS-UFSC
Campus Universitário - Trindade - Florianópolis - SC
Caixa Postal 476 - CEP 88040-900
e-mail: {merkle,fraga,lau}@lcmi.ufsc.br

Resumo

Este artigo apresenta um esquema de autorização para redes de larga escala, envolvendo modelos e ferramentas de programação representados pelo Web, Java e CORBA. O esquema de autorização é fundamentado em estruturas e conceitos introduzidos no Web, Java e CORBA, com relação a segurança. Um protótipo desse esquema é apresentado nesse texto, onde as soluções adotadas visando a sua concretização são discutidas. A implementação visa políticas discricionárias e não discricionárias. Essas políticas são fundamentadas em modelos de segurança importantes da literatura como o Matriz de Acesso, Bell e Lapadula e modelos *Role-based Access Control Models*. Esses modelos são adaptados ao contexto que definimos no sentido de elaborar um esquema de autorização realizável, prático onde seja fácil se definir políticas a serem seguidas em sistemas distribuídos de larga escala.

Palavras-Chave: Segurança, Políticas de autorização, Esquemas de autorização.

1. Introdução

Novos serviços estão surgindo para atender a demanda crescente e as novas necessidades em sistemas distribuídos de larga escala como a *Internet*. A segurança (*Security*) sempre foi uma questão muito importante nesses sistemas, muito embora tenha sido um pouco desconsiderada inicialmente na evolução de redes de larga escala e aplicações distribuídas em geral. Com o aumento do uso dessas redes em aplicações críticas como o comércio eletrônico, a segurança passou a se firmar como um requisito básico nesses sistemas.

Por outro lado, essa crescente demanda de serviços e novas aplicações nesses sistemas têm provocado o surgimento de novos paradigmas e ferramentas de programação distribuída. De acordo com essa tendência, aparece também uma nova geração de aplicações distribuídas, cuja característica distinta é a exploração da mobilidade de código. Essas novas abordagens introduzem também novos problemas de segurança e também implicam na necessidade de novos conceitos e modelos de segurança diante desse novo quadro. Entre esses novos paradigmas e ferramentas estão, além dos códigos móveis, o próprio Web e o CORBA. Esquemas de autorização para esses tipos de sistemas envolvendo mobilidade de código, portanto, devem sofrer um processo de renovação e amadurecimento, o que fornece um amplo tema de pesquisa na área de segurança.

A linguagem Java popularizou o conceito de código móvel através dos seus *applets* executados a partir de *browsers* Web. O ambiente Web representa a estrutura mais simples para códigos móveis, disponibilizando toda uma rede mundial e possibilitando carga de códigos em qualquer ponto da rede. O modelo de programação distribuída Java/CORBA/Web está se tornando padrão *de facto* de programação na Internet [1] [2]. A combinação da carga automática do código do cliente e, também, a completa independência de plataforma operacional é uma grande vantagem das aplicações Java/CORBA/Web.

O objetivo deste artigo é apresentar a proposição de um esquema de segurança para aplicações distribuídas em redes de larga escala envolvendo esses novos modelos de programação representados pelo Web, Java e CORBA. O esquema de autorização é fundamentado em estruturas e conceitos introduzidos no Web, Java e CORBA, com relação a segurança, e em modelos de segurança importantes da literatura como o Matriz de Acesso, o modelo Bell e Lapadula [3] e os modelos Baseados em Papéis (*Role-based access control models* [4]). Esses modelos são adaptados ao contexto definido para oferecer soluções para os problemas de segurança encontrados no desenvolvimento de aplicações distribuídas seguras.

Nesse artigo, a seção 2 apresenta os conceitos de política de segurança e de esquema de autorização. Na seção 3, as ferramentas Java, Web e o CORBA são examinadas segundo seus conceitos, modelos e suportes visando a segurança. O esquema de autorização proposto para redes de larga escala, combinando os modelos de segurança Java/CORBA/Web e considerando políticas de autorização discricionárias é apresentado na seção 4. Os resultados de implementação conseguido com um primeiro protótipo são descritos na seção 5. Trabalhos relacionados são resumidos na seção 6. Na última parte do artigo são feitas conclusões sobre os resultados obtidos e são apontadas algumas perspectivas futuras.

2. Modelos de Segurança

O conceito de *segurança* em um sistema informático é identificado com a sua capacidade de assegurar a prevenção do acesso e a manipulação ilegítimas da informação ou, ainda, de evitar a interferência indevida na sua operação normal [5]. Esta capacidade está fundamentada sobre quatro propriedades que devem ser mantidas: confidencialidade, disponibilidade, integridade e autenticidade. Assegurar a segurança de um sistema é então uma tarefa colossal diante das dimensões dos sistemas atuais.

O conjunto de propriedades de segurança que se deseja assegurar em um sistema e também a maneira como são garantidas essas propriedades estão fortemente vinculadas à definição de política de segurança do sistema. A *política de segurança* ou *política de autorização* de um sistema é o conjunto de regras e direitos que determinam a maneira pela qual as informações e os outros recursos são gestionados, protegidos e distribuídos no interior de um sistema específico. As políticas de autorização são geralmente descritas fazendo uso de modelos de segurança [6]. Modelos importantes da literatura que podem ser citados são o modelo Matriz de Acesso, Bell e Lapadula e os Baseados em Papéis. As políticas se classificam em duas categorias: políticas discricionárias e políticas obrigatórias ou não-discricionárias.

No caso das políticas *discricionárias*, os direitos de acesso a cada recurso, são manipulados livremente pelo responsável do recurso (geralmente o proprietário do mesmo), segundo a sua vontade (à sua discrição). O modelo matriz de acesso, introduzido por Lampson [6], descreve políticas de autorização discricionárias e está fundamentada sobre as noções de sujeitos, objetos e direitos. Um sujeito tem o direito de acesso sobre um objeto podendo executar a operação correspondente sobre o objeto, se esse direito estiver expresso na matriz de acesso.

As políticas ditas *obrigatórias* ou não-discricionárias resumem em seus esquemas de autorização, um conjunto de regras incontornáveis que expressam um tipo de organização envolvendo a segurança das informações no sistema como um todo. A política obrigatória supõe os usuários e os objetos ou recursos do sistema, todos etiquetados; as etiquetas dos objetos seguem uma classificação específica enquanto os usuários ou os sujeitos do acesso possuem níveis de habilitação. Os controles que determinam as autorizações de acesso são baseados numa comparação da habilitação do usuário com a classificação do objeto. As regras definidas nesses controles e que são ditas como incontornáveis asseguram que o sistema verifique as propriedades de confidencialidade e de integridade. Os modelos baseados sobre treliças descrevem políticas de autorização não-discricionárias. Um dos exemplos importantes de modelos baseados sobre treliças é o modelo Bell e Lapadula do DoD (*Department of Defense* – Departamento de Defesa dos Estados Unidos) [6].

Os esquemas de políticas obrigatórias são utilizados conjuntamente com os discricionários. Nesse caso, um usuário é autorizado a manipular uma informação se possui o direito ao acesso correspondente (controle discricionário) e se é habilitado ao nível de classificação da informação (controle obrigatório). As políticas discricionárias e obrigatórias são reconhecidas em padrões oficiais como a classificação de sistemas seguros do DoD chamado TCSEC (*Trusted Computer System Evaluation Criteria*) também conhecido como Livro Laranja (*Orange Book*) [4].

2.1 Esquemas de autorização em sistemas distribuídos

Um *esquema de autorização* é a concretização da política de autorização através de um conjunto de mecanismos. Esses mecanismos asseguram que todos os acessos a objetos no sistema são autorizados pela política definida. No que se refere a controle de acesso, esses mecanismos são implementados por um conjunto de recursos de *hardware* e *software* formando o que poderíamos chamar de *núcleo de segurança*. Outros controles internos são também importantes na concretização de uma política de autorização como os controles criptográficos, serviços de autenticação, serviços de identificação etc. Esses controles adicionais formam o que o TCSEC identifica como base informática de confiança (TCB – *Trusted Computing Base*). O núcleo de segurança e o TCB são duas noções introduzidas pelo TCSEC como essenciais para a construção de sistemas seguros.

Se considerarmos sistemas distribuídos podemos afirmar que existem abordagens centralizadas na implementação de esquemas de autorização e abordagens baseadas na distribuição de funções de segurança no sistema. Dentre as abordagens baseadas na distribuição de funções de segurança do sistema encontram-se as abordagens Kerberos [5], Delta-4 [7] e X.509 [8]. Na abordagem Kerberos a autenticação do sistema é gerida por um servidor único confiável (o Kerberos), enquanto a autorização é controlada independentemente por cada sítio do sistema. Essa abordagem apresenta dificuldades em manter a coerência da política de autorização visto que a matriz de acesso se apresenta nesse caso

particionada envolvendo todos os sítios da rede (vários servidores decidindo concorrentemente sobre os acessos no sistema). O servidor único de autenticação é um ponto vulnerável do sistema.

Na abordagem Delta-4 é definido um *quorum* de sítios de autenticação, chamados na literatura original de sítios de segurança [7]. Nessa abordagem, um cliente tem acesso a objetos em servidores desde que consiga a autenticação em uma maioria dos servidores de segurança. Os vários sítios de segurança que formam o *quorum* na autentificação são também responsáveis pela gestão dos acessos a objetos persistentes no sistema.

Se considerarmos redes de larga escala, como a Internet, um controle de acesso a objetos persistentes fica difícil em um nível global. As funções de nível global nesses sistemas se limitam a um servidor de autenticação normalmente na forma hierarquizada (X.509). Na abordagem X.509, padrão ITU-T [8], o serviço de autenticação é particionado no sentido de aumentar a sua aplicabilidade em redes de larga escala como a Internet. Diferentes autoridades de certificação (CA – *Certification Authority*) são estruturadas na forma de uma árvore. Um cliente particular é registrado em um desses CA's. O acesso de um cliente a objetos de um servidor registrado em outro CA, implica na interação entre os CA's do cliente e do servidor no processo de autenticação.

3. Mecanismos de Segurança nas Ferramentas Web, Java e CORBA

3.1 Web

O modelo de programação distribuída fornecido a partir do ambiente WWW é extremamente poderoso, escalável e útil no desenvolvimento de aplicações distribuídas. A existência de um ambiente desse porte que seja seguro é necessário quando se têm objetivos de utilizá-lo nas mais diversas aplicações em um contexto de redes de larga escala [9].

A evolução do Web tem acrescentado novas características à esse ambiente para satisfazer a crescente demanda, sem considerar de forma cuidadosa, pelo menos inicialmente, o impacto na segurança do sistema. De uma maneira geral, o problema de segurança do Web pode ser dividido em três partes: segurança do servidor, segurança da informação em trânsito e segurança do cliente.

A segurança do servidor está fundamentada em serviços de autenticação e de controle de acesso. Os serviços de autenticação disponíveis são o esquema básico, o esquema com *digest authentication* e o esquema com certificados [9]. O esquema básico de autenticação é um sistema de identificação baseado no fornecimento de um nome de usuário e de uma senha. Não existe no mesmo uma preocupação em fornecer alguma forma de controle que dê garantias da validade dessas informações. O esquema designado *digest authentication* fornece alguns mecanismos de troca permitindo autenticação dessas informações por meio de um *digest* calculado sobre essas informações de identificação. Na autenticação em sistemas distribuídos de larga escala, é desejável a intermediação entre os pares comunicantes de uma entidade confiável (autoridade de certificação) que, através de emissão e revogação de certificados, garanta as trocas de informações na autenticação de um cliente perante um servidor. Existem duas formas de controlar o acesso ao servidor Web: negando o acesso a um cliente que deseja conexão com o servidor com base no seu endereço IP, ou proibindo o acesso a um cliente até que ele produza alguma forma de identificação, tipicamente um nome de usuário e a senha correspondente. Nessa segunda abordagem, uma vez conectado o usuário fica sujeito à proteção normal de diretórios através de listas de acesso.

A segurança do WWW também depende da segurança das informações que trafegam através da Internet. A proteção dessas informações no suporte de comunicação envolve o que poderíamos chamar de controles criptográficos. Os protocolos criptográficos disponíveis para uso na Internet que podem ser citados são: o SSL (*Secure Socket Layer*) e o PGP (*Pretty Good Privacy*) [9].

Os clientes (*browsers*) também podem sofrer ataques. Linguagens e ambientes de programação para o Web como Java e Javascript, com suas novas características impostas para melhorar a interatividade do ambiente WWW, também criaram problemas de segurança adicionais. A segurança do cliente é abordada no contexto da linguagem Java.

3.2 Java

Neste artigo, *código móvel* se refere ao *software* que viaja através de uma rede heterogênea, atravessando domínios de proteção e sendo executado automaticamente no seu destino. A segurança representa um grande problema nos sistemas que fornecem suporte à mobilidade de código e freqüentemente é considerada a principal limitação para o amplo uso desses paradigmas [10]. A linguagem Java popularizou a noção de código móvel através dos seus *applets*. O modelo de segurança implementado pela plataforma Java, na sua proposição inicial, é centrada sobre o conceito de

sandbox [11]. A essência do modelo *sandbox* é que o código local é confiável e tem acesso completo aos recursos do sistema (como o sistema de arquivos) enquanto o código remoto (um *applet*) não é confiável e pode acessar apenas recursos limitados, fornecidos dentro do *sandbox*. Esse conceito de *sandbox* é empregado pelo *toolkit* de desenvolvimento Java (*Java Development Kit* – JDK) e é geralmente adotado pelas aplicações construídas a partir do JDK, incluindo os *browsers* Web habilitados a executar código Java.

A segurança na plataforma Java é concretizada em vários níveis. Primeiramente, a linguagem que é interpretada, é projetada para ser segura em relação aos tipos de dados utilizados. Outra parte importante desse modelo de segurança está nas ações do compilador e do verificador de *bytecodes* que garantem apenas a execução de códigos Java legítimos. Ao ser compilado, o código já sofre uma verificação de tipos minuciosa. Porém, como os *browsers* que carregam os arquivos de classes não sabem se os *bytecodes* correspondentes foram produzidos por um compilador Java confiável, um verificador de *bytecodes* (*bytecode verifier*) é ativado para inspecionar esse código antes de sua execução sobre a máquina virtual (*Java Virtual Machine* - JVM). O verificador de *bytecodes* representa um mecanismo de controle de acesso estático baseado na inspeção de código.

Também uma peça importante na segurança Java é o carregador de classes (*class loader*), uma ferramenta programável que fornece os meios para recuperar e ligar, de forma dinâmica, as classes de uma aplicação em uma JVM, fornecendo assim a idéia de mobilidade da linguagem. O *class loader* é chamado pela JVM em execução quando o código que está executando contém uma referência não resolvida a um nome de uma classe. O *class loader* recupera a classe correspondente, possivelmente de um *host* remoto e então carrega a classe na JVM. Nesse ponto o código correspondente é executado na máquina virtual (JVM).

Ao ser executado, o acesso aos recursos do sistema é mediado pela JVM, através de uma classe chamada *Security Manager* (gerente de segurança), que restringe as ações de um código não confiável ao mínimo possível. Esse mecanismo implementa o controle de acesso (dinâmico) na execução dos códigos, concretizando a idéia do *sandbox*.

Duas fases de evolução podem ser citadas a partir do modelo de segurança *sandbox* original [11]. A primeira envolve o *kit* JDK 1.1.x que introduziu o conceito de *applet* assinado. Nesse modelo, um *applet* assinado digitalmente é tratado como se fosse um código local correto, desde que a chave da assinatura seja reconhecida como confiável pelo sistema que recebeu o *applet* para execução. Além do conceito de *applet* assinado, JDK 1.1.x definiu uma nova API, chamada de *Java Cryptography Architecture API*, que foi projetada para fornecer aos desenvolvedores de aplicações o acesso a funcionalidades criptográficas na plataforma Java [12]. No *kit* JDK 1.1, essa "arquitetura de funções criptográficas" incluiu classes e interfaces para prover assinaturas digitais e *message digests*.

A segunda evolução do modelo de segurança do Java é apresentada no *kit* JDK 1.2 onde estão presentes os seguintes objetivos: prover controle de acesso de grão fino, possibilitar políticas de segurança facilmente configuráveis, definir uma estrutura de controle de acesso que pode ser estendida facilmente para todos os programas Java, incluindo aplicações e *applets*. O conceito de domínios de proteção é fundamental nessa nova arquitetura, cumprindo em parte esses objetivos citados. Um domínio pode ser definido como o conjunto de objetos que são diretamente acessados por um principal[1]. Domínios de proteção possuem duas categorias distintas: domínios do sistema e domínios da aplicação. É importante que todos os acessos aos recursos externos, tais como sistema de arquivos, serviços de rede, tela e teclado sejam acessíveis apenas através dos domínios do sistema.

Uma política de segurança do sistema, definida pelo usuário ou pelo administrador do sistema, deve especificar quais domínios de proteção devem ser criados e quais permissões devem ser fornecidas nesses domínios. O ambiente de execução Java mantém um mapeamento do código (classes e instâncias) para seus domínios de proteção e permissões correspondentes. O conceito de domínio implementa de certa forma o princípio de menos privilégio.

No JDK 1.2, é também feita uma extensão à API criptográfica no sentido de suportar certificados X.509v3 [13].

---

1 Um *principal* é definido na literatura de *security* como um usuário, processo ou qualquer entidade ativa registrada e autenticável no sistema.

3.3 CORBA

As especificações CORBA/OMG correspondem a um conjunto de padrões e conceitos para objetos distribuídos em ambientes abertos propostos pela OMG (*Object Management Group*)[2]. Esses padrões contribuem para que métodos de objetos remotos possam ser invocados, de forma transparente, em ambientes distribuídos heterogêneos através de um ORB (*Object Request Broker*). O ORB, num sentido mais genérico, é um canal de comunicação para objetos distribuídos.

As especificações CORBA definem características de um ambiente de suporte à aplicações distribuídas segundo a arquitetura OMA (*Object Management Architecture*). Essa arquitetura divide o espaço de objetos distribuídos em três partes: objetos de aplicação, construídos pelo programador de aplicação; objetos de serviço (COSS: *Common Object Services Specification*) que suportam serviços comuns, independentes de domínios de aplicação e ainda, em facilidades comuns (*Common Facilities*) que por sua vez, são serviços orientados para domínios de aplicações. Todos os objetos se comunicam nessa arquitetura via ORB.

A OMG redigiu um documento definindo as principais linhas no que se refere a segurança de objetos distribuídos [14]. Os serviços de segurança do CORBA formam o volume 3, fazendo parte dos objetos de serviço (Serviços COSS) definidos pelo CORBA. O modelo de segurança descrito nesse documento estabelece vários procedimentos envolvendo a autenticação e a verificação da autorização na invocação de um método remoto, segurança da comunicação entre os objetos, além de aspectos envolvendo 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: objetos de aplicação (nível de aplicação), objetos de serviço, serviços ORB e o núcleo do ORB (todos a nível de *middleware* CORBA), componentes de tecnologia de segurança (a nível de serviços de segurança subjacentes) e componentes de proteção básica, fornecidos por uma combinação de *hardware* e sistemas operacionais locais. A figura 1 ilustra os níveis e componentes principais do modelo CORBA de segurança, indicando os relacionamentos entre eles. Nessa figura, os objetos cliente e destino representam o nível das aplicações.

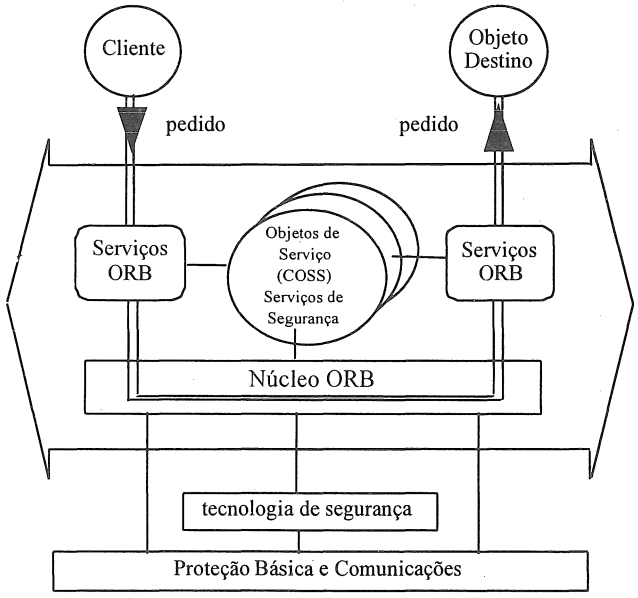

Fig. 1. Modelo CORBA de segurança.

Serviços ORB e objetos de serviço (serviços COSS) são construídos sobre o núcleo ORB e estendem as funções básicas com qualidades ou controles adicionais, facilitando a implementação de objetos distribuídos. Uma combinação de serviços ORB e serviços COSS são usadas na implementação da segurança no modelo CORBA. Nas especificações do

[2] 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.

modelo de segurança CORBA, os chamados serviços ORB tomam os nomes de *interceptors*. Logicamente, um *interceptor* é interposto no caminho de uma chamada entre um cliente e um destino. Cada serviço COSS relacionado com a segurança é associado a um *interceptor*, cuja finalidade é provocar o desvio transparente ao objeto de serviço correspondente.

No modelo CORBA de segurança são definidos dois *interceptors* que atuam durante uma invocação de método: o *Access Control Interceptor* que em nível mais alto provoca um desvio para realizar o controle de acesso na chamada e, o *Secure Invocation Interceptor* que faz uma interceptação de mais baixo nível no sentido de fornecer propriedades de integridade e de confidencialidade nas transferências correspondentes à invocação. Esses interceptadores atuam, tanto no cliente como no objeto de aplicação servidor. No interior do maior retângulo da figura 2, estão contidos o *Access Control Interceptor que* está representado pelo objeto *Controle de Acesso* e o *Secure Invocation Interceptor que* está representado pelo objeto *Chamada Segura*. Os interceptadores que atuam em invocações de um método são criados durante o processo de *binding* (ligação) entre dois objetos de aplicação que devem se comunicar através dessas invocações.

Os objetos de serviço que implementam os controles de segurança nas especificações CORBA são o *Principal Authenticator* que corresponde ao serviço de autenticação de principais no CORBA, o *Vault* que estabelece as associações seguras entre os clientes e servidores com os respectivos contextos de segurança, o *Credential* que representa as credenciais ou direitos do cliente na sessão. Existem outros objetos de serviço relacionados com a não-repudiação e auditoria.

A autenticação descrita nas especificações de segurança do CORBA, define um conjunto de trocas entre o principal e o objeto de serviço *Principal Authenticator*. Essas trocas tem como objetivos finais a aquisição de credenciais pelo principal. Um principal autenticado pode adquirir os privilégios que serão necessários para que ele possa acessar os objetos do sistema. Esses privilégios estão contidos em uma *credencial* (objeto *Credential*). Depois de adquirir credenciais, o principal e os objetos que atuam em seu nome, como clientes, podem iniciar chamadas a objetos servidores.

Os *interceptors* que atuam nas invocações de métodos são criados durante o processo de *binding* entre dois objetos de aplicação que se comunicam durante essas invocações. Distinguimos a *invocação inicial* quando ocorre o processo de ligação entre objetos de uma *invocação normal* que representa qualquer chamada entre objetos feita depois que a ligação foi estabelecida. Uma ligação cria o contexto para a comunicação segura entre as partes comunicantes e é iniciada pela execução do cliente de uma operação *binding*. O objeto *Controle de Acesso* é criado nos lados do cliente e do servidor no sentido de executar o controle de acesso em ambos os lados quando da invocação de um método. O objeto *Chamada Segura* é o responsável em tempo de ligação pelo estabelecimento da associação segura entre o cliente e o servidor. Em tempo de ligação (*binding*), esse interceptador de baixo nível ativa o objeto de serviço *Vault* no sentido de criar um objeto *Contexto de Segurança* da associação segura que deve ser estabelecida entre os objetos cliente e servidor. O objeto *Contexto de Segurança* retém informações de chaves, algoritmos criptográficos, etc, usados nessa associação.

Os objetos de serviço no modelo CORBA de segurança, na verdade, isolam aplicações e o ORB da tecnologia de segurança (figura 1), que 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. Várias tecnologias podem ser utilizadas para fornecer esses serviços: SSL, SPKM (*Simple Public-Key GSS-API Mechanism*), Kerberos e CSI-ECMA [14]. Essa tecnologia de segurança pode ser acessada via interfaces de segurança genéricas tais como a GSS-API (*Generic Security Services API*), que isolam as implementações dos serviços de segurança dos detalhes funcionais dos serviços subjacentes.

4. JaCoWe – Nossa Proposição de Autorização para Redes de Larga Escala Integrando os Modelos Java, CORBA e Web

A integração Java/CORBA/Web constitui um ambiente poderoso para a programação de aplicações distribuídas em redes de larga escala. Essa combinação de ferramentas forma um *Web de objetos distribuídos* que permite que se explore a flexibilidade e a disponibilidade fornecida pela tecnologia WWW: a máquina do usuário, não importando em que ponto se encontra na rede mundial, necessita apenas de um *browser* para carregar o código cliente. O usuário pode não estar conectado a um servidor remoto, mas simplesmente interagir ou ativar o *software* cliente carregado pelo *browser* em seu computador. O *software* cliente pode fornecer uma interface gráfica (GUI) que aceita pedidos do usuário e faz o *display* de informações. Nesse modelo o servidor remoto (servidor CORBA) fornece um conjunto de serviços que pode variar do

simples acesso a informação até a execução de códigos remotamente. Na integração dessas ferramentas, a linguagem Java contribui com o suporte para códigos móveis e o CORBA com a tecnologia de integração de objetos interoperáveis.

A princípio, nesse ambiente, aplicações distribuídas são expressas na forma de clientes representados por códigos móveis ou *applets* Java, e de servidores implementados como objetos distribuídos. O usuário quando apresenta a URL do serviço desejado, provoca a carga no *browser* do *applet* residente na máquina servidora Web. A execução do código do *applet* pode conter chamadas a métodos remotos que são tratados pelo servidor CORBA na máquina servidora. O cliente (*applet* Java) interage com um servidor CORBA (objeto remoto) através de um ORB, em um modelo cliente/servidor tradicional[3].

O *esquema de autorização* proposto é para ser utilizado no contexto de aplicações distribuídas em redes de larga escala. Ele está fundamentado nas estruturas e conceitos introduzidos no Web, Java e CORBA com relação a segurança. Nesse esquema de autorização são definidos dois níveis de controle de segurança: o *nível global* e o *nível local*. Esses dois níveis são concretizados em objetos de serviço CORBA e por núcleos de segurança e TCB's, respectivamente. Os objetos de serviço concentram funções de identificação e autenticação de usuários e os controles de autorização no acesso de objetos visíveis a nível global. Os núcleos de segurança e TCB's, presentes em cada máquina do sistema, validam os acessos aos recursos locais. A figura 2 mostra os componentes principais que implementam os controles desse esquema.

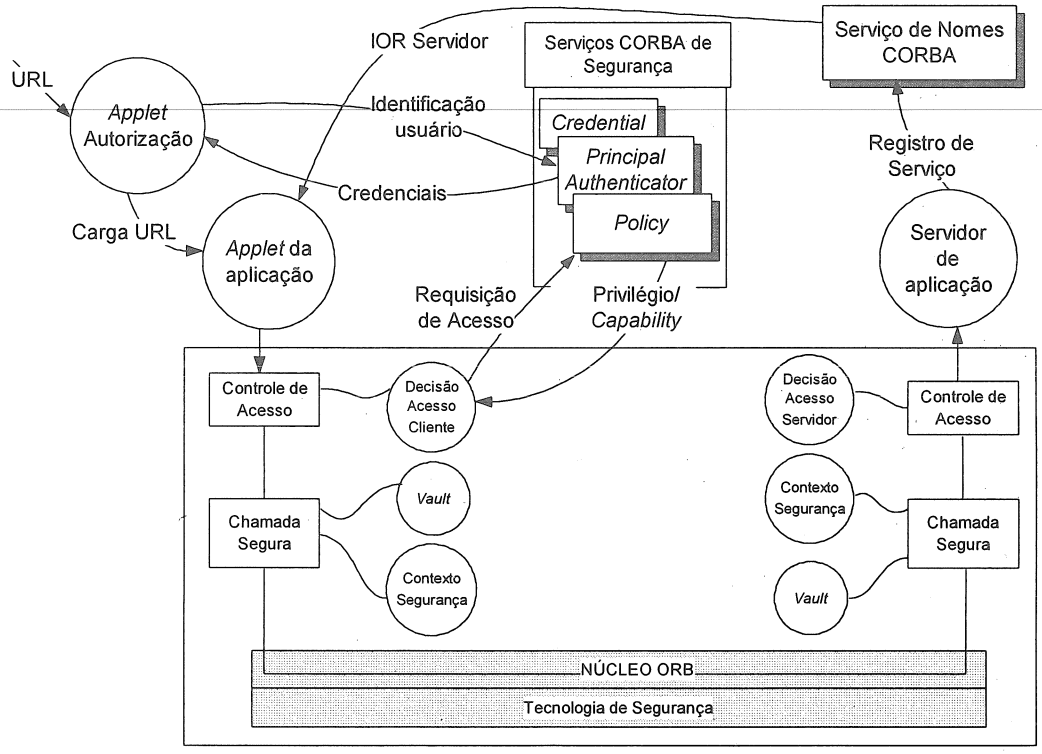

Fig. 2. Estrutura do Esquema de Autorização *JaCoWe*.

[3] *Applets* clientes estão sujeitos às restrições impostas pelo *browser* nos quais se executam. Essas restrições impedem que esses códigos possam ter acesso ao disco local ou fazer uma conexão com um *host* diferente daquele a partir do qual foram carregados. Isso de certa maneira impõe a localização de objetos servidores de aplicação na máquina do servidor Web a partir da qual o *applet* cliente foi carregado. Algumas técnicas são oferecidas para permitir que os *applets* clientes se libertem dessas restrições e possam se comunicar com objetos em sistema distribuído sem limitação de localização. Nós adotamos a do *Wonderwall* [ION97b], solução proprietária da Iona, que se constitui em um *software* localizado na estação do servidor Web, atuando como um roteador de mensagens do *applet* para o servidor de aplicação. Nesse caso, mensagens IIOP do *applet* são transferidas na máquina do servidor Web aos objetos de destino em outras estações.

A autenticação de usuários ou principais nesse esquema é feita usando um *applet* e um objeto de serviço. O *applet* de autenticação (código móvel) interage com o objeto de serviço *PrincipalAuthenticator* segundo as trocas especificadas pelo CORBA para a identificação e autenticação de principais. O usuário quando apresenta a URL do serviço desejado, provoca a carga desse *applet* de autenticação. Uma vez verificada a identificação do usuário, estabelecendo as credenciais CORBA do cliente (objeto de serviço designado *Credential*), o *applet* de autenticação libera a URL correspondente, carregando o *applet* de aplicação (cliente CORBA).

O *applet* de aplicação, inicialmente, interage com o serviço de nomes CORBA (*CosNaming*) [15], para obter, a partir do nome do objeto, a referência ou IOR (*Interoperable Object Reference*) do objeto servidor de aplicação. Isso deve permitir a ligação com o objeto servidor. As chamadas executadas por esse *applet* de aplicação sobre um servidor remoto da aplicação estão sujeitas a dois níveis de controle de acesso. No nível mais alto, um objeto de serviço CORBA designado *Policy* é responsável pela validação de pedidos de acesso a objetos persistentes, verificando os direitos em uma lista de acesso, segundo a política apropriada. A partir dessa verificação de alto nível são geradas *capabilities* (competências) que serão validadas localmente nos servidores remotos, completando então o segundo nível de controle de acesso definido em nosso esquema.

Para montar esses dois níveis de controle de acesso, utilizamos os dois níveis de interceptação definidos no modelo CORBA de segurança, designados na figura 2 como objetos *Controle de Acesso* e *Chamada Segura*. A interceptação de alto nível (*Controle de Acesso*), no lado do cliente (*applet* de aplicação), desvia para o objeto *Policy* para as verificações de lista de acesso. No lado do servidor da aplicação, essa interceptação de alto nível é usada para validar a *capability* recebida com a requisição de invocação. A validação de uma *capability* é realizada em um objeto de serviço local que faz parte do TCB da máquina onde se localiza o servidor de aplicação.

A interceptação de baixo nível (*Chamada Segura*) do modelo CORBA, em ambos os lados (cliente e servidor), é usada para proteger em uma associação segura a mensagem que transporta a requisição com a *capability*. A representação de privilégios (ou direitos) na forma de *capabilities* no esquema de autorização se utiliza de estruturas CORBA, tais como referências de objetos e credenciais. Além dos controles citados acima, os controles criptográficos também são necessários no esquema e são definidos também na forma de objetos de seviço CORBA que usam a tecnologia de segurança residente sob o ORB (objetos *Vault* e *Contexto de Segurança*).

Os núcleos de segurança e *Trusted Computing Bases* validam os acessos locais dos *applets* de aplicação. Para implementar essa base confiável que valida os acessos locais, usamos o modelo de segurança Java e seus procedimentos de controle de acesso. O modelo de segurança Java, na sua versão 1.2, possui arquivos de política que identificam quais operações podem ser realizadas pelos códigos carregados (*applets*). Além disso, conta com o gerente de segurança que realiza o controle de acesso baseado na política concretizada através dos domínios de proteção. Para proteger a máquina hospedeira de um código móvel (*applet*), é necessário passar por uma fase de autenticação do código móvel e, durante a execução do mesmo, validar os acessos a recursos locais. A autenticação serve para garantir a procedência do código. Os privilégios retornados (credenciais) quando da autenticação do usuário determinam a construção do domínio de proteção associado ao *applet* de aplicação ativado pelo usuário. Esses domínios de proteção determinam a dependência desses acessos locais em relação às políticas globais.

As interações do objeto *Controle de Acesso* e do *applet* de autenticação com os objetos de serviço *Policy* e *PrincipalAuthenticator*, respectivamente, se dão através do ORB, usando associações seguras previamente definidas. As submissões de URL de um usuário podem se dar através do protocolo *https*, estabelecendo o uso do SSL e seus controles criptográficos durante as interações com o servidor Web.

Em redes de larga escala a autorização passa, necessariamente, pela conexão de vários servidores de nomes, cada um responsável por um domínio específico de objetos e usuários. Em cada domínio de nome, os controles do esquema de autorização devem se fazer presentes, centralizados em objetos de serviço próprios ao domínio considerado. Ou seja, cada domínio de nomes deverá ter os seus objetos *Policy* e *PrincipalAuthenticator* centralizando os controles globais sobre os objetos persistentes e usuários do domínio. As especificações X.500 fornecem meios para essas conexões entre diferentes contextos de nomes baseadas em mecanismos de *alias* [8]. O *alias* pode ser entendido como uma designação local em um domínio, identificando um objeto ou usuário como não local e que permite a procura na resolução de nomes estendida a outros domínios. Inicialmente, o esquema de autorização compreende um domínio único

mas, para viabilizá-lo em redes de larga escala, vislumbra-se a possível utilização do LDAP (*Lightweight Directory Access Protocol*) que é uma implementação de serviços de diretórios baseada no X.500 e bastante utilizada atualmente [16] [17].

5. Resultados de Implementação

Um primeiro protótipo do *JaCoWe* foi desenvolvido em nossos laboratórios. As implementações são feitas usando as ferramentas OrbixWeb 3.0 da Iona [15], Netscape Communicator 4.5, JDK 1.2 da Sun e o SSLeay - versão *free* do protocolo SSL 3.0 [18]. O SSL (*Secure Socket Layer*) foi escolhido como tecnologia de segurança por apresentar código fonte disponível e livre para ser modificado, por ser um protocolo amplamente utilizado em aplicações na Internet e por estar inserido pela OMG em sua última revisão do serviço de segurança [14] como um dos possíveis protocolos a serem utilizados por aplicações que implementam a segurança do CORBA. No protótipo foi usado o protocolo SSLeay 0.9.0b.

Essa primeira experiência implementa políticas discricionárias, se limitando a uma única verificação de controle de acesso no lado do servidor de aplicação. Esse protótipo se limita também a um só domínio de nomes. A figura 3 sintetiza as funcionalidades implementadas no protótipo.

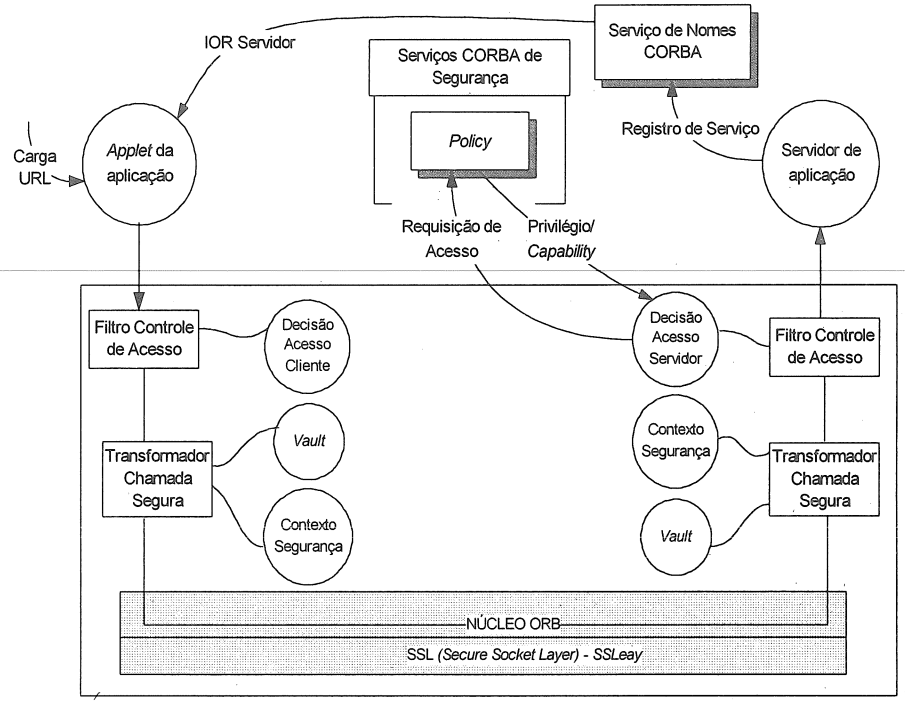

Fig. 3. Estrutura do Protótipo do Esquema de Autorização.

Dentre os objetos implementados nesse protótipo, além dos objetos *Applet de aplicação* e *Servidor de aplicação*, temos os objetos de *Decisão de Acesso*, *Vault*, *Contexto de Segurança* e *Policy* que são objetos do modelo CORBA de segurança. Esses objetos utilizam outros objetos de serviço (COSS) CORBA como o serviço de nomes.

O controle de acesso realizado no protótipo está baseado em um mecanismo de lista de acesso implementado no objeto *Policy*. As entidades que estão sujeitas aos controles de segurança no nosso sistema são os usuários e os objetos. A lista se resume aos usuários e os seus direitos de execução dos métodos nos objetos. O objeto *Policy* que armazena a estrutura da lista de acesso tem a descrição de sua interface apresentada em IDL CORBA na figura 4.

A lista de acesso é composta por identificações de objetos cada qual com suas listas de usuários e direitos. O direito do usuário é expresso através de uma identificação e da lista de métodos aos quais o usuário tem direito de acesso (estrutura *UserRight*). É possível realizar a inserção de um novo direito de acesso em um objeto específico (método *insertRight*), a remoção dos direitos de um usuário (método *removeRight*) ou de todos os direitos de um objeto (*removeAllRight*), a atualização dos direitos de acesso de um objeto (*updateRight*) e a obtenção dos métodos aos quais o sujeito terá direito de acessar e que irão formar a *capability* (*provideCapability*). Os métodos de inserção, atualização e remoção de direitos de acesso são executados pelos proprietários dos objetos servidores de aplicação, logo após terem sido registrados no serviço de nomes.

No sentido de facilitar as interações no protótipo, o objeto *Policy* foi colocado juntamente com o servidor de aplicação CORBA e com o servidor Web em uma mesma estação. Com isso o controle por *capabilities* na estação do servidor, presente no modelo da seção anterior, se torna desnecessário (o controle de nível global é feito na própria estação). Embora nessa montagem o objeto *Policy* já ofereça os métodos para a montagem de *capabilities*.

```
Interface Policy {
        exception NotFound {};
        exception CannotProceed {};
        exception InvalidName {};
        exception AlreadyInserted {};

        typedef sequence<string> MethodList;

        struct UserRight {
                string user;
                MethodList ml;
        };

        typedef sequence<UserRight> UserRightList;

        boolean insertRight ( in string objectname, in UserRightList url )
                raises ( NotFound, CannotProceed, InvalidName, AlreadyInserted );
        boolean removeRight ( in string objectname, in string user )
                raises ( NotFound, CannotProceed, InvalidName );
        boolean removeAllRight ( in string objectname )
                raises ( NotFound, CannotProceed, InvalidName );
        boolean updateRight ( in string objectname, in UserRightList url )
                raises ( NotFound, CannotProceed, InvalidName );
        MethodList provideCapability ( in string objectname, in string user )
                raises ( NotFound, CannotProceed, InvalidName );

};
```

Fig. 4. Especificação IDL do objeto *Policy*.

As implementações realizadas no protótipo usam como mecanismos de desvios os *filtros* (*filters*), presentes no OrbixWeb, em substituição aos *interceptors* do modelo de segurança CORBA, os quais ainda não foram completamente definidos pela OMG. No protótipo, o único filtro de controle de acesso atuante é o colocado na estação do servidor de aplicação. Esse filtro invoca o objeto *DecisãoAcessoServidor* que, por sua vez, interage com o objeto *Policy* para as verificações de lista de acesso na máquina do servidor.

Outro tipo de interceptador presente no OrbixWeb é o *transformador* (*transformer*) que diferentemente do filtro, pode operar sobre os pedidos. No protótipo, *transformadores* são usados na interceptação de baixo nível. Durante a ligação entre cliente e objeto servidor, na estação do cliente, o *transformador Chamada Segura* ativa o objeto de serviço *Vault* no sentido de criar o objeto *ContextoSegurança*, estabelecendo a associação segura. O objeto *ContextoSegurança* executa o *handshake* SSL entre cliente e servidor para ter posse das informações relativas à associação estabelecida.

Durante uma invocação normal, na estação do cliente, o *transformador Chamada Segura* ativa o objeto *ContextoSegurança* no sentido de obter informações de chaves, algoritmo criptográfico, etc, usados na associação. Essas informações serão passadas ao *transformador* para executar o processo de cifragem do SSL sobre a mensagem IIOP, antes de transmiti-la via TCP. No lado do servidor, o objeto transformador, de posse das informações do objeto *ContextoSegurança* requisita ao SSL a realização da decifragem da mensagem.

O objeto *Principal Authenticator* (figura 2), responsável pela autenticação dos principais e pela criação do objeto *Credential*, nesse protótipo inicial também não foi implementado. As credenciais nesse protótipo são criadas estaticamente. Em relação a autenticação de *applets*, os mecanismos correspondentes também não foram implementados. Esses mecanismos de assinatura e de verificação deverão ser facilmente concretizados a partir do uso da API criptográfica do JDK 1.2 [13].

O mecanismo de *capabilities* no nosso esquema está sendo especificado. A concretização desse mecanismo deverá fazer uso do *Request* enviado pelo cliente quando de uma invocação de método remoto. Como um *Request* é composto pelo IOR do objeto servidor e de um campo indicando o método solicitado, a simples adição dos direitos do usuário forma o *ticket* caracterizando então uma *capability*, que no seu conceito mais simples contém o nome do objeto receptor e os direitos do possuidor desse *ticket* sobre o objeto. O filtro de controle de acesso provê meios na modificação de uma requisição (objeto *Request*). Dessa forma, no fluxo de execução de uma chamada, depois que o objeto *DecisãoAcessoCliente* obteve do objeto *Policy* os direitos referentes a chamada, esses serão inseridos pelo filtro de controle de acesso no *Request*. A *capability* será cifrada no *transformador* antes do seu envio, bem como todos os campos

importantes do *Request*. No lado do servidor, o transformador decifra os dados, recuperando a *capability* para que a mesma possa ser validada pelo objeto *DecisãoAcessoServidor*. Para evitar ataques com mensagens antigas são necessários mecanismos para incluir campos de *nonce* na *capability*. Esses *nonces* podem ter sido negociados, durante o estabelecimento da associação segura e estarem disponíveis a partir do objeto *ContextoSegurança*. A presença de um mecanismo de *capabilities* fará com que nosso esquema retome a forma apresentada na figura 2, onde o objeto *Policy* reside em máquina diferente do servidor de aplicação e é ativado a partir da estação do cliente.

6. Trabalhos relacionados

Na literatura, em geral, são poucas as experiências explorando os modelos e conceitos de segurança introduzidos nas especificações CORBA e no Java. Bem mais raros devem ser os estudos envolvendo a integração dos conceitos dessas ferramentas. A própria OMG faz uma única referência em [19], sobre a combinação do conceito de mobilidade de código com o modelo CORBA de segurança. A idéia nesse documento é estender a suportes de agentes móveis os controles disponíveis em serviços de segurança CORBA.

Uma experiência acadêmica envolvendo segurança e o CORBA pode ser encontrada em [20]. Nesse trabalho é feita uma proposta de um modelo de segurança para objetos distribuídos suportados pelo CORBA. Os principais serviços de segurança tratados no artigo são autenticação do cliente e servidor, controle de acesso e serviços de proteção. Nesse modelo, o controle de acesso é concretizado na forma de um monitor de referência distribuído que examina cada pedido de acesso de cliente com informações de controle do objeto servidor. Esse monitor de referências constitui uma camada entre o núcleo ORB e suas interfaces, e é chamado de *Object Security Controller* (OSC). Nesse modelo é introduzida a noção de nós *ORB*, que são máquinas compartilhadas por objetos e clientes. Cada nó *ORB* armazena informações de controle de seus objetos na forma de listas de acesso. As informações de controle são disponíveis de forma global a partir de um serviço de nomes. O artigo discute a sua proposta assumindo mecanismos de autenticação baseados em chave pública . Também são apresentadas formas de criação de objetos dinâmicos e informações de segurança. Essa experiência não está baseada nas especificações CORBA para segurança. O particionamento das informações de segurança (lista de acesso, chaves, etc) é diferente de nossa abordagem e nos parece bem mais vulnerável.

Existem alguns produtos implementando o modelo CORBA de segurança que podem ser citados [21] como: o OrbixSecurity da empresa Iona Technologies [22], o DAIS da empresa PeerLogic e o ORBAsec SL2 da empresa Adiron. A exceção do OrbixSecurity que implementa o nível 1 do modelo de segurança CORBA, todos os outros seguem o nível 2 que fornece um conjunto mais completo de funcionalidades. Esses produtos adotam os padrões *GSS-API* e o Kerberos como tecnologia de segurança subjacente. O controle de acesso nesses produtos é feito somente no servidor de aplicação. Isso dificulta o uso desses produtos em nossos propósitos de implementações de políticas globais. Existem outros produtos e softwares disponíveis que combinam o CORBA com o SSL. OrbixSSL da Iona, Visibroker SSL da Inprise, ORBacus SSL da empresa OOC e suporte SSL do ORB MICO [23] são exemplos dessa combinação.

7. Considerações Finais

A combinação da carga automática do código do cliente e, também, a completa independência de plataforma operacional tornam a integração das ferramentas Java, CORBA e Web altamente desejáveis para a programação de objetos distribuídos na Internet. O objetivo deste artigo foi o de apresentar a proposição de um esquema de segurança para aplicações distribuídas em redes de larga escala. Esse esquema de autorização foi concebido no sentido de ser realizável, prático e onde se possa definir políticas a serem seguidas em aplicações distribuídas de larga escala. O esquema apresentado está fundamentado em estruturas e conceitos introduzidos no sentido da segurança nas ferramentas citadas.

Nesse esquema de autorização são definidos dois níveis de controle de segurança: o nível global e o nível local. Esses dois níveis são concretizados por objetos de serviço CORBA e por núcleos de segurança e TCB's locais. Os objetos de serviço concentram funções de identificação e autenticação de usuários e os controles de autorização ao acesso de objetos visíveis a nível global. Os núcleos de segurança e TCB's, presentes em cada máquina do sistema, validam os acessos dos *applets* aos recursos locais.

Os objetos de serviço CORBA são concebidos no sentido de residirem em máquinas não compartilhadas e seguras da rede. Uma possibilidade é a localização no *site* do servidor Web, considerando o mesmo um *host* seguro da rede ou do domínio de nomes considerado. A centralização de informações de controle nesses objetos de serviço, definindo o nível

global do esquema de autorização, minimiza o impacto na violação de um núcleo de segurança ou de uma outra máquina do sistema considerado.

Um protótipo está sendo construído no sentido de verificar a viabilidade do esquema de autorização proposto. Na sua versão atual, esse protótipo se apresenta bastante simplificado. Nessa versão, os objetos de serviço, os objetos de aplicação e o servidor Web compartilham a mesma máquina. Nessa primeira implementamos apenas políticas discricionárias, tomando como base uma única verificação de controle de acesso no lado do servidor de aplicação. Esse protótipo, com os testes realizados, serviu para nos dar garantias em relação a certas opções de implementação do esquema. Basicamente nos limitamos nessas implementações em concretizar os mecanismos de controle nas interações entre *applets* de aplicação e objetos servidores de aplicação. Os mecanismos de segurança que atuam durante essas interações foram concebidos para serem totalmente transparentes à aplicação.

Um dos objetivos do esquema de autorização é que permita de maneira simples a implementação de políticas de segurança. Políticas discricionárias são simples e de certa maneira foram implementadas nos experimentos realizados até o momento. Políticas não-discricionárias ou obrigatórias são objetivos futuros em nosso protótipo. Usando os objetos *Credential* e *Policy* do esquema e mais o mecanismo de domínio introduzido no *kit* JDK 1.2 do Java acreditamos que possamos implementar uma versão do modelo Bell e Lapadula ou ainda de modelos *Role-based Access Control Models*.

Referências Bibliográficas

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

[2] G. Lea, "Microsoft Finally Consents to Marriage of Software Models," OMG News, 1998. http://www.omg.org/news/glea.html

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

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

[5] V. Nicomette, "La Protection dans les Systèmes à Objets Répartis,". *PhD thesis*, Institut National Polytechnique de Toulouse, 1996.

[6] C. E. Landwehr, "Formal models for computer security," *ACM Computing Surveys*, vol. 13, no. 3, pp. 247-278, Sep. 1981.

[7] J. S. Fraga, "La Sécurité des Donnés par la Tolérance aux Intrusions," *PhD thesis*, Institut Nat. Polytechnique de Toulouse, 1985.

[8] ITU-T. "Information Technology - The Open Systems Interconnection - The Directory: Autentication Framework," *ITU-T Recommendation X.509*, Nov. 1993.

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

[10] T. Thorn, "Programming Languages for Mobile Code," *ACM Computing Surveys*, vol. 29, no. 3, pp. 213-239, Sep. 1997.

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

[12] Sun Microsystems Inc., "Java Cryptography Architecture API Specification & Reference," Sun Microsystems Inc., May 1997.

[13] Sun Microsystems Inc., "Java Cryptography Architecture API Specification & Reference," Sun Microsystems Inc., Oct. 1998.

[14] OMG, "Security Service:v1.2 Final," *OMG Document Number 98-01-02,* in CORBAservices: Common Object Services Specification, Nov. 1998.

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

[16] D. Kosiur, "LDAP: The next-generation directory?,", *SunWorld Online*, Oct. 1996. http://www.sunworld.com/swol-10-1996/swol-10-ldap.html

[17] T. Burghart, "CORBA as an LDAP Server Datastore - An architecture for intranet directory services", *IDM (Internet Design Magazine)*, 1998. http://idm.internet.com/features/corba-ldap.shtml

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

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

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

[21] U. Lang, "Current State of CORBA Security in Practice – CORBA Security Service Implementations and other CORBA Security Products," *University of Cambridge – Computer Laboratory*, 1998. http://www.cl.cam.ac.uk/~ul201/research.html

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

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

[24] C. M. Westphall, "Esquemas de Autorização para Programação Distribuída combinando os Modelos de Segurança Java/CORBA/Web," *Exame de Qualificação de Doutorado*, LCMI-DAS-UFSC, Certificado de Registro no. 165.863, livro 277, folha 4 da Fundação Biblioteca Nacional, Ministério da Cultura, Escritório de Direitos Autorais, Rio de Janeiro, Brazil, Aug. 1998.

[25] Q. Zhong and N. Edwards, "Security Control for COTS Components," *IEEE Computer*, vol. 31, no. 6, June 1998.