PoliCap - Um Serviço de Política para o Modelo CORBA de Segurança

Carla Merkle Westphall*, Joni da Silva Fraga e Michelle Silva Wangham
Universidade Federal de Santa Catarina - LCMI-DAS-UFSC
Campus Universitário - Trindade - Florianópolis - SC - Brasil
C.P. 476 - CEP 88040-900
e-mail: {merkle, fraga, wangham}@lcmi.ufsc.br

Resumo - Este artigo propõe o *PoliCap* - um Serviço de Política a ser utilizado por aplicações distribuídas que utilizam o modelo CORBA de segurança. O *PoliCap* foi projetado para ser inserido no contexto do projeto de segurança *JaCoWeb*. Esse projeto está desenvolvendo um esquema de autorização com o objetivo de concretizar a gerência de políticas de segurança em redes de larga escala como a Internet. O serviço de política *PoliCap* supre a carência existente na utilização de políticas de segurança para a programação com objetos distribuídos. O artigo ainda apresenta os resultados de uma implementação realizada e a avaliação dos mesmos usando o Critério Comum de Segurança, padrão ISO 15408.

Palavras-chave - Segurança, Políticas de segurança, Esquema de autorização, CORBA.

Abstract - This paper presents *Policap* - a Policy Service to be utilized for distributed applications that use CORBA security model. *Policap* was proposed for insertion in the *JaCoWeb* Security Project. This project is developing an authorization scheme in order to deal with management of security policies in large-scale networks, such as the Internet. The *Policap* policy service fills in the existing gap in the use of security policies for distributed object programming. The paper further presents the implementation resuits obtained and an evaluation of these resuits based on Common Criteria, ISO standard 15408.

Keywords - Security, Security Policies, Authorization Scheme, CORBA.

1 INTRODUÇÃO

Atualmente, negócios tornam-se cada vez mais dependentes de seus sistemas de informação e, portanto, necessitam também que essas informações sejam acessadas de forma correta e segura. Nos sistemas distribuídos de larga escala, a proteção da informação é uma tarefa difícil de ser realizada já que a proteção física dos computadores que contêm informações estratégicas dificilmente é atingida. Além disso, como a informação é compartilhada, tecnicamente, qualquer pessoa pode ter acesso ao sistema através da rede.

A gerência de políticas de segurança em sistemas de larga escala como a Internet é uma questão muito importante atualmente. Entretanto, apresenta várias dificuldades devido a existência de um grande número de usuários, objetos e operações, a falta do uso de políticas de segurança e ambientes heterogêneos presentes em redes de larga escala. Dessa forma, a complexidade e a escalabilidade dificultam a gerência das políticas de segurança [1].

A complexidade inerente aos sistemas de objetos distribuídos causa vulnerabilidades adicionais que devem ser contidas pela arquitetura de segurança. Dentre estas vulnerabilidades, o controle de acesso em sistemas de larga escala é problemático, já que sistemas de objetos distribuídos podem crescer sem limites e componentes são constantemente adicionados, removidos e modificados nestes ambientes. Em sistemas geograficamente muito grandes normalmente existem diferentes domínios de políticas de segurança, o que dificulta suas administrações.

No sentido de minimizar estes problemas a OMG *(Object Management Group)*[1] introduziu o *CORBAsec* que é um modelo de segurança que, quando implementado e administrado corretamente, pode prover um alto nível de segurança para informações e aplicações de objetos distribuídos em ambientes de larga escala.

Segundo este modelo, quando um objeto é criado e é feito visível via CORBA em um sistema, automaticamente ele também torna-se membro de um ou mais domínios de políticas de segurança. Entretanto, as propostas incluídas no modelo são recentes e alguns problemas são facilmente identificados: as especificações atuais *[2]* não cobrem procedimentos para gerenciar membros dos domínios e não definem procedimentos para incluir ou remover políticas *[2]* [3].

---

* Doutoranda (bolsista da CAPES) do Curso de Pós-Graduação em Engenharia Elétrica da UFSC.

[1] É um consórcio de empresas com o objetivo de especificar um conjunto de padrões e conceitos para a programação orientada a objetos em ambientes distribuídos abertos.

relação a cada objetivo de segurança, em conformidade com todos os outros requisitos definidos em [16], pode-se definir que o protótipo estende o *CAPP* no sentido de usar uma política de segurança discricionária, sendo considerado um TOE nível EAL3. Isso quer dizer que o protótipo é metodicamente testado e verificado, ou seja, o protótipo é aplicável a circunstâncias onde se necessita um nível moderado de segurança em um TOE convencional.

6 TRABALHOS RELACIONADOS

A literatura apresenta alguns trabalhos sobre gerência de políticas de segurança em ambientes de larga escala. A própria especificação do serviço de gerência de domínios [3] do CORBAsec não está padronizada. Um projeto acadêmico que pode ser mencionado é o *Cherubim* [17]. O *Control* [18] é um produto que implementa o CORBAsec e a gerência de políticas de segurança.

A arquitetura de segurança do projeto *Cherubim* consiste de duas partes: uma IDL CORBA estendida com o objetivo de automatizar a *ligação* entre políticas de segurança e aplicações e, um *framework* de segurança para agentes dinâmicos. Seu principal objetivo é construir um sistema de segurança dinâmico capaz de prover mecanismos de segurança específicos para aplicações em um ambiente móvel e de larga escala. O controle de acesso nesse projeto foi desenvolvido usando-se o conceito de *capabilities*, que carregam os direitos de acesso do *principal* até a máquina do servidor onde ocorre o processo de autorização. O projeto elaborou interfaces para manipular políticas de autorização discricionárias, mandatórias *(MAC)* e baseadas em papéis *(RBAC)* mas não utiliza os objetos *COSS* do *CORBAsec* implementando objetos próprios. O ORB utilizado é o *JacORB* [5] e os serviços criptográficos são implementados com a API criptográfica do *Java IAIK* [13].

O *Control* [18] é um ORB que estende o *ORBAsec* com os serviços *COSS* do *CORBAsec* que implementam serviços de autenticação, transmissão segura de mensagens e controle de acesso automático. A política de autorização discricionária é estabelecida com uma linguagem de controle de acesso usada por objetos servidores para definir seus objetos de política. A interceptação do controle de acesso é executada no lado do objeto servidor. A gerência de políticas de segurança é feita usando-se o módulo de gerência de domínios.

Comparando o *PoliCap* com as experiências apresentadas, verificamos que o *PoliCap é* um serviço que realiza o controle de acesso *no lado do cliente,* ao contrário das propostas descritas. O *PoliCap* combina características do *Cherubim,* como as *capabilities,* e do Control, com o adicional do uso restrito de objetos padronizados ou em via de padronização no CORBAsec.

7 CONCLUSÕES

O *PoliCap* preenche uma lacuna no gerenciamento de políticas de segurança existente no modelo CORBAsec, concretizando o primeiro nível de controle de acesso do projeto *JaCoWeb.* O segundo nível de controle de acesso é desenvolvido com o uso das *capabilities* propostas para o modelo CORBAsec. O *PoliCap* e as *capabilities* para o CORBAsec fornecem uma contribuição importante para gerenciar políticas de autorização em redes de larga escala. Importante também é a descrição do funcionamento dinâmico relativo aos objetos de serviço e interceptadores do CORBAsec.

O protótipo é uma experiência real de implementação do modelo discricionário do CORBAsec, considerando que existem poucas experiências desse tipo disponíveis. Esse protótipo facilita a gerência e administração de políticas de segurança, permitindo que a segurança seja garantida a nível de ORB. A segurança garantida a nível de ORB tem vantagens como transparência da segurança para aplicações e maior confiança nos serviços de segurança que são sempre executados.

TOE e de seus requisitos e testes funcionais. Para tanto, desenvolveu-se um ST para o protótipo, que é chamado *ST-Protótipo-JaCoWeb*. Esse ST atua em conformidade com o [16] atendendo o nível EAL3 do CC.

O *ST-Protótipo-JaCoWeb*, como definido para um ST, é composto por uma descrição do TOE, pelas ameaças às quais o TOE está sujeito, pelas políticas de segurança, por objetivos de segurança que determinam os requisitos funcionais que devem existir no sistema e uma declaração da conformidade a algum PP disponível e certificado (como o *CAPP)*.

A descrição do TOE já foi apresentada na seção 4. As ameaças (*Threats*) 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.

Dentre os tipos de políticas de segurança existentes no protótipo estão a política discrionária designada *P.DAC (Discretionary Access Control)* que visa impedir a ocorrência das ameaças acima citadas. Os direitos de acesso aos recursos e informações fornecidos pelo protótipo estão disponíveis através dos objetos *DomainAccessPolicy* e *RequiredRights*.

Os objetivos de segurança são determinados a fim de estabelecer os requisitos funcionais necessários para conter as ameaças definidas para um sistema e concretizar a política de segurança estabelecida. Dentre os objetivos de segurança definidos que estão em conformidade com o *CAPP* estão os seguintes: *O.AUTHORIZATION*, que estabelece que as funções de segurança do protótipo devem ser implementadas de forma que apenas usuários autorizados tenham acesso ao sistema, *O.DISCRETIONARY_ACCESS*, que estabelece que as funções de segurança do protótipo devem fornecer permissões de acesso aos recursos e informações baseado nos atributos de privilégio dos sujeitos, *O.MANAGE*, que estabelece que o protótipo deve fornecer suporte ao administrador autorizado do sistema e *O.ENFORCEMENT*, que estabelece que o protótipo deve ser projetado e implementado de forma aplicar as políticas de segurança no lado do objeto servidor da aplicação.

Cada objetivo de segurança é implementado por um conjunto de requisitos funcionais definidos para o protótipo. Como o protótipo está sendo avaliado em conformidade com [16], pode-se assumir o mesmo conjunto de requisitos funcionais e de garantia de segurança. Apenas como exemplo, o objetivo *O.DISCRETIONARY_ACCESS é* implementado pelos seguintes requisitos de segurança: FDP_ACC.1 (política de controle de acesso discricionária, implementada pelos objetos do CORBAsec), FPD_ACF.1 (função de controle de acesso implementada pelo objeto *ServerAccessDecision*), FIA_ATD.1 (definição de atributos do usuário feita de forma estática no protótipo), FIA_USB.1 (ligação entre usuário e principal que é estabelecida caso o usuário seja *gerente* ou *cliente* do servidor bancário do protótipo), FMT_MSA.1 (gerência dos atributos de segurança dos objetos, implementada pelo objeto *RequiredRights* do CORBAsec) e FMT_MSA.3 (inicialização de atributos estáticos, implementado para gerenciar os objetos de política no protótipo).

Verificando cada um dos requisitos funcionais e de garantia existentes no protótipo, com

carregados, livrando os *applets* das restrições impostas pelo modelo *sandbox* do Java, permitindo comunicação com quaisquer objetos sem limite de localização. Dessa forma o servidor Web e o servidor da aplicação não necessitam permanecer na mesma máquina.

O objeto *PrincipalAuthenticator*, responsável pela autenticação e criação do objeto *Credential*, não foi implementado nesse primeiro protótipo. As credenciais do protótipo são criadas estaticamente e possuem atributos de segurança que identificam os direitos do cliente no sistema e o mecanismo de segurança que está sendo utilizado (SSL, no nosso caso).

O *PoliCap* e as *capabilities* representam uma evolução do protótipo inicial. A presença desses dois componentes fará com que o nosso modelo retome o seu formato original, como apresentado na figura 2, onde a autorização é também realizada na máquina do cliente.

5 AVALIAÇÃO DO PROTÓTIPO USANDO O CRITÉRIO COMUM DE SEGURANÇA

O Critério Comum (*Common Criteria - 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*. Esse critério deriva de padrões anteriores como TCSEC (ou livro laranja), ITSEC, CTCPEC e conta com a colaboração do Canadá, França, Alemanha, Holanda, Inglaterra e Estados Unidos. Tomou-se padrão ISO 15408 em 1999 [14].

O CC [14] define como expressar os requisitos de segurança de um sistema ou produto, define categorias distintas de requisitos funcionais (*functionalrequirements*) 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.

Alguns conceitos importantes desse critério compreendem o TOE, PP e ST. O *objeto de avaliação (TOE - Target of Evaluation) é* a parte do produto ou sistema que está sujeita à avaliação, no nosso caso, o TOE a ser considerado é o protótipo desenvolvido. O *profile de proteção (PP - Protection Profile) é* uma definição de conjuntos de requisitos e objetivos, independentes de implementação, que permite aos consumidores e desenvolvedores criar conjuntos padronizados de requisitos de segurança que estejam de acordo com suas necessidades. No caso da avaliação do protótipo realizada foi utilizado um PP registrado e aceito como padrão, designado *ControlledAccess Protection Proflle (CAPP)* [16], que define requisitos e objetivos relativos a um objeto de avaliação e utiliza, dentre outras características, políticas discricionárias. As ameaças de segurança ao TOE, os objetivos, os requisitos e a especificação resumida das funções e garantias de segurança, juntos, formam as entradas principais ao *alvo de segurança (ST — Security Target).* O ST contém os objetivos de segurança e requisitos de um TOE específico e define as medidas funcionais e de garantia oferecidas por esse TOE para preencher os requisitos determinados. O ST pode declarar conformidade a um ou mais PPs *e forma a base para uma avaliação.*

O CC fornece garantia de segurança de um TOE usando o conceito de *investigação ativa* que é uma avaliação de um sistema ou produto de tecnologia da informação a fim de *determinar suas propriedades de segurança.* Técnicas de avaliação que podem ser usadas são: análise do projeto do TOE e de seus requisitos, testes funcionais, análise de vulnerabilidades, dentre outras. A exemplo de critérios anteriores, o CC também possui um conjunto de níveis de garantia de segurança. Os níveis EALs (*EvaluationAssurance Leveis)* uniformizam o modo de balancear a garantia obtida com o custo e possibilidade de adquirir tal grau de garantia. Existem sete níveis de garantia: EAL1, EAL2, EAL3, EAL4, EAL5, EAL6 e EAL7.

As técnicas utilizadas para avaliar o protótipo desenvolvido foram: análise do projeto do

único domínio de nomes. A figura 5 sintetiza as funcionalidades implementadas no protótipo.

Dentre os objetos implementados nesse protótipo (além do *applet* e servidor da aplicação) estão os objetos *ServerAccessDecision*, *Vault*, *SecurityContext*, *DomainAccessPolicy*, *RequiredRights*, *SecurityManager* e *PolicyCurrent* do modelo CORBA de segurança. Esses objetos usam outros serviços CORBA (COSS) como o serviço de nomes.

O controle de acesso executado pelo protótipo é baseado no mecanismo de lista de acesso implementado pelos objetos *DomainAccessPolicy* e *RequiredRights*. As entidades que estão sujeitas aos controles de segurança no nosso sistema são os principais (com seus atributos de privilégio) e os objetos servidores da aplicação (com seus atributos de controle).

O objeto *ServerAccessDecision* é responsável pela validação dos pedidos de acesso aos métodos do objeto servidor, interagindo com os objetos *DomainAccessPolicy* e *RequiredRights*. O método *access_allowed* (figura 6) do objeto *ServerAccessDecision* obtém os direitos requeridos invocando o método *get_required_rights* do objeto *RequiredRights* e obtém os direitos fornecidos pela política *DomainAccessPolicy* invocando o método *get_effective_rights*. Compara os direitos requeridos e os direitos fornecidos ao atributo de privilégio para decidir se o método a ser invocado pode ou não pode ser executado.

```
// Método access_allowed do objeto ServerAccessDecision
public boolean access_allowed(Org.omg.SecurityLevel2.Credentials[] cred_list, Org.omg.CORBA.Object target, String
operation_name, String target_interface_name) {

boolean b = false;
// Obtém os direitos requeridos -- invoca get_required_rights
Org.omg.Security.RightsListHolder rightslist = new Org.omg.Security.RightsListHolder();
Org.omg.Security.RightsCombinatorHolder combinator = new Org.omg.Security.RightsCombinatorHolder();
rights.get_required_rights((Org.omg.CORBA.Object) null, operation_name, "sistemaBancario.idl", rightslist, combinator);

//extensibleFamily do projeto JaCoWeb - ver classe SSLCredentials
Org.omg.Security.ExtensibleFamily extFamily = new Org.omg.Security.ExtensibleFamily((short) 0, (short) 1);
Org.omg.Security.AttributeType[] attributes = new Org.omg.Security.AttributeType[1];
attributes[0] = new Org.omg.Security.AttributeType(extFamily, 5);

// Obtém os direitos fornecidos pelo objeto DomainAccessPolicy -- invoca get_effective_rights
granted = access.get_effective_rights(cred_list[0].get_attributes(attributes), extFamily);
for (int i = 0; i < access.getCount(); i++) { // Compara direitos requeridos e fornecidos para permitir ou negar o acesso
                    if (granted[i] != null) {
                                if ((granted[i].right).equals(rightslist.value[0].right)) {
                                            i++; b = true; break; }
                    }           }           return b;}
```

Figura 6. Método *access_allowed* do objeto *ServerAccessDecision*.

Os objetos *DomainAccessPolicy* e *RequiredRights* residindo somente na mesma máquina do servidor da aplicação (controle de acesso apenas no lado do servidor) torna desnecessário o uso de *capabilities* em um segundo nível de controle. Esses objetos são criados estaticamente, *em tempo de ligação* entre os objetos de aplicação comunicantes. A inserção, atualização e remoção de direitos nesses dois objetos são executadas pelos próprios objetos servidores da aplicação, logo depois de se registrarem no serviço CORBA de nomes (*CosNaming*).

A implementação usa como mecanismos de desvio os *interceptors* presentes no JacORB. No protótipo, o único interceptador de controle de acesso é criado na máquina do servidor de aplicação. O *interceptador de controle de acesso* invoca o objeto *ServerAccessDecision* que, por sua vez, interage com os objetos *RequiredRights e DomainAccessPolicy* para a autorização na máquina do servidor.

No protótipo, foram utilizadas as funções criptográficas IAIK-JCE e o pacote iSaSiLk v.5.2 que implementa o SSL v3 em Java (disponível em [13]). Toda a negociação e uso do SSL, no protótipo, é executada de forma transparente pelo iSaSiLk [19].

A *assinatura digital* do modelo de segurança do Java foi utilizada para permitir aos clientes acesso ao disco local ou conexões com máquinas diferentes a partir das quais foram

credencial do cliente. Depois das inclusões para gerar a *capability*, o *Request* fica representado como na figura 4.

Request novo com a *capability* (IOR, método, emissor, *nonce*)					
Request antigo			Campos adicionados		
IOR servidor	Método solicitado	...	Identificador Emissor	*Nonce*	*Random*

Figura 4. Estrutura do *Request* com campos adicionais para implementar a *capability*.

Em *tempo de decisão de acesso*, o interceptador de controle de acesso *no lado do servidor* faz a verificação da *capability*. Dessa forma pode-se construir *capabilities* para cada uma das invocações de um cliente no *CORBAsec*. Essas *capabilities* são utilizadas para formar um segundo nível de controle de acesso no esquema de autorização *JaCoWeb*.

4 RESULTADOS DE IMPLEMENTAÇÃO

Um protótipo - incluindo o serviço de políticas *PoliCap* e o mecanismo de *capabilities* - foi desenvolvido em nossos laboratórios (figura 5). Uma aplicação exemplo constituída de um sistema bancário composto por um objeto servidor CORBA e um *applet* cliente Java foi construída para testar as implementações deste protótipo. O objeto servidor CORBA foi desenvolvido com a ferramenta JacORB 1.0 [5], um ORB Java *free*, e o *applet* cliente foi implementado com a ferramenta JDK 1.2.1. O *browser* Netscape Communicator 4.5 também foi usado como ambiente para a interação entre o cliente e o servidor. O objetivo dessa implementação foi de concretizar uma política de controle de acesso discricionária baseada nas estruturas definidas no *CORBAsec*.

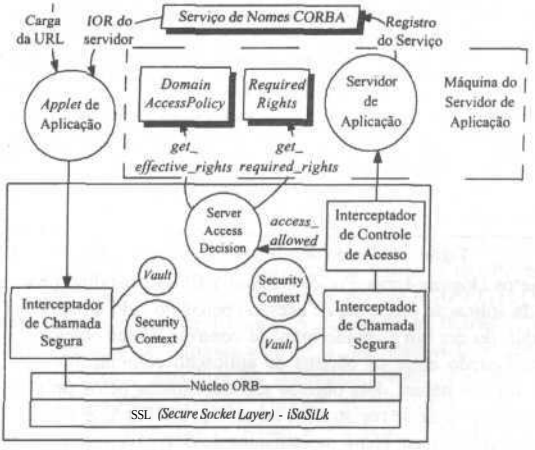

Figura 5. Estrutura do Protótipo Implementado.

Essa versão do *PoliCap* preocupou-se em amadurecer primeiramente, todo o aspecto dinâmico dos objetos de serviço do CORBAsec, o que não é uma tarefa trivial. Isso porque a especificação utilizada é extremamente ampla [2] e o dinamismo de uma aplicação que usa o CORBAsec não é descrito na literatura, portanto, requer um esforço considerável e prevê o entendimento de um grande número de objetos. Portanto, foi feito um experimento que não comporta ainda os objetos *DomainAccessPolicy* e *RequiredRights locais*, estabelecidos *em tempo de ligação* entre cliente e servidor.

O experimento implementa políticas discricionárias, estando limitado a uma única verificação de controle de acesso *no lado do servidor*. O protótipo também está limitado a um

primeira requisição. As requisições *de* operações subseqüentes na interface do servidor serão validadas apenas com o mecanismo de competências. Os dois níveis de controle de acesso reduzem o tráfego de rede em caso da negação de acesso e ao mesmo tempo a segurança do sistema não depende unicamente da integridade do cliente.

O mecanismo de competências não é padronizado pelo *CORBAsec*, embora exista a sugestão sobre o uso de *capabilities* em [2], a partir das abstrações criadas no modelo. Não se pode afirmar que exista uma definição universal *do conteúdo* de uma *capability*. Tradicionalmente, uma *capability* deve conter a referência de um objeto e um conjunto de direitos. A *capability* define os direitos do detentor sobre o objeto em questão. Em [8], uma *capability* clássica é representada como a tripla *(IdObject, Rights, Random)* contendo o nome do objeto, um conjunto de direitos de acesso e um número aleatório, respectivamente. O número aleatório previne falsificação, sendo normalmente resultado de uma função *one-way f*, aplicada sobre o identificador do objeto e os direitos sobre o mesmo *(Random = f(IdObject, Rights))*[3]. Quando uma requisição chega no lado do servidor, com a sua respectiva *capability*, a função *one-way f* é executada novamente e o seu resultado é conferido com o número aleatório enviado para detectar possíveis modificações *na* mensagem *da* requisição (*tampering*). Este número aleatório que faz o papel de um *nonce*, assegura também a propriedade de '*freshness* f 10] da requisição do cliente.

Assim como não pode ser possível falsificar uma *capability*, também não deve ser possível reutilizá-la ou repassá-la a terceiros. Para isto, normalmente, além do número aleatório é incluído em uma *capability* um campo de identificação do solicitante da operação (o cliente). A *capabilities* são protegidas por mecanismos de cifragem quando transmitidas no suporte de comunicação [8].

No esquema *JaCoWeb*, as *capabilities* são criadas dinamicamente a cada requisição do cliente para operações no servidor. A classe *Request* (estrutura do pedido) definida nas especificações CORBA *é* usada na composição de uma *capability* no nosso esquema. Uma *capability* no *JaCoWeb* contêm nos seus campos: a *IOR (InteroperableObject Reference)do objeto servidor*, representando a identidade da entidade sobre a qual a *capability* fornece os direitos de acesso; o *método solicitado*, representando o direito; o *identificador do emissor/principal*, representando a identidade da entidade solicitante da operação; e um *nonce* que assegura a propriedade '*freshness* do pedido. Em um *Request* de uma invocação usual no CORBA, *o IOR* do objeto servidor e a identificação do método solicitado já estão presentes. Os outros dois campos da *capability*, identificador do emissor do pedido e um campo de *nonce*, devem ser inseridos no *Request* durante a interceptação de alto nível, no objeto *AccessDecision* do lado do cliente. O valor de *nonce* é calculado da seguinte forma:

nonce = f (identificador do emissor, método solicitado, IOR do servidor, Random);

onde / corresponde à função *hash one-way* SHA, presente no pacote *java.security* [11]. O valor *Random* é um número aleatório calculado fazendo uso da classe *java.security.SecureRandom* da API Criptográfica do Java JDK 1.2 [l 1]. Este valor aleatório é uma redundância que garante a propriedade *freshness* da mensagem do *Request*, garantindo contra os ataques por mensagens antigas [8]. Os valores de *Random* são inseridos nas mensagens de *Request* e enviados para o servidor funcionando como um contador de mensagens. Todos os pedidos de operações do cliente sobre a interface do servidor, terão as mensagens correspondentes enviadas pela associação segura, seguindo a numeração de seqüência *Random, Random+1, Random+2,...* .

O identificador do emissor (ou solicitante da operação) é conseguido a partir da

---

[3] Existem vários algoritmos práticos que implementam funções *one-way hash*. Os algoritmos de MD4, MD5 e SHA são exemplo destas funções [9].

autorização definida e o objeto de *política* correspondente com o domínio de *política*. A operação *delete_policy* da *interface* *DomainAccessPolicyAdmin* remove a *política* de autorização (e o objeto de política correspondente) do domínio.

A operação *get_local_domain_policy* da *interface* *DomainAccessPolicyAdmin* monta um objeto *DomainAccessPolicy* com os direitos efetivos (*GrantedRights*)presentes na política de autorização relativas ao atributo de privilégio e os seus estados de delegação (sujeito). Na verdade esta operação monta localmente, *em tempo de ligação,* uma versão do *DomainAccessPolicy,* essencial na validação *(em tempo de decisão de acesso)* de diversas operações existentes em uma mesma interface. Por exemplo, tomando o objeto *DomainAccessPolicy* com as informações da tabela 1 - objeto global definido no domínio -, ao se executar a operação:

```
localDomainAccessPolicy = get_local_domain_policy (
                          SecClientInvocationAccessDiscretionary,'role,autoridade,caixa', initiator),
```

o retorno desta operação é o objeto *DomainAccessPolicy* - uma versão representada na tabela 3 - que deve atuar localmente no objeto *AccessDecision* de uma invocação.

Atributo de Privilégio	Estado de **Delegação**	**Direitos** Fornecidos (***GrantedRights***)
role:caixa	Initiator	corba: g--u

Tabela 3. Objeto *DomainAccessPolicy* retornado pela operação *get_local_domain_policy*.

A operação *get_local_required_rights* da *interface RequiredRightsAdmin* monta também uma versão local, com os direitos requeridos presentes no objeto *RequiredRights* global (centralizado) do serviço de política do domínio. Esta versão do objeto *RequiredRights,* montada localmente *em tempo de ligação,* contém todas as linhas relativas a uma interface, considerando que um objeto cliente pode executar as diversas operações definidas nesta interface de objeto de aplicação. Por exemplo, tendo-se o objeto *RequiredRights* (global) da tabela 2, ao se executar a operação:

```
localRequiredRights = get_local_required_rights (Renda_fixa),
```

o valor de retorno da operação é o objeto *RequiredRights* (local) representado na tabela 4.

Direitos Requeridos (***Required Rights***)	**Combinador de Direitos**	**Operação**	**Interface**
Corba:g	All	**Ver_Saldo**	**Renda_Fixa**
Corba:gs	Any	**Depositar**	**Renda_Fixa**

Tabela 4. Objeto *RequiredRights* retornado pela operação *get_local_required_rights*.

3.3 *Capabilities* usando o *CORBAsec*

No modelo *CORBAsec,* as decisões de controle de acesso podem ser realizadas tanto no lado do cliente como no lado do servidor. O controle de acesso no lado do objeto destino é definido nas especificações do *CORBAsec* como *normal.* Mas, também é salientado que quando as verificações de controle de acesso são feitas no lado do cliente, as negações de acesso ao objeto servidor determina uma redução no tráfego de rede. Nota-se também que em alguns ORB's, considerações de integridade do sistema podem tomar indesejável a confiança no controle de acesso feito unicamente no lado do cliente [2].

No *JaCoWeb,* o objetivo é realizar o controle de acesso aos objetos distribuídos visíveis via CORBA, assumindo que, num primeiro nível as verificações são realizadas pelo objeto *AccessDecision* do cliente em parceria com o serviço *PoliCap.* A execução das operações *get_local_domain_policy*e *get_local_required_rights* a partir da interceptação de controle de acesso *em tempo de ligação,* monta os objetos *DomainAccessPolicy* e *RequiredRights* locais. Estes objetos fazem parte da estrutura necessária no objeto *AccessDecision* do cliente para gerar *capabilities* (competências) a serem verificadas no objeto *AccessDecision* do servidor. A verificação de alto nível só é feita uma vez, durante a *ligação* dos objetos cliente e servidor, na

disponíveis através dos objetos *DomainAccessPolicy* e *RequiredRights*. As interfaces destes objetos permitem o acesso de políticas discricionárias. *Aplicações administrativas* são responsáveis pela definição desses objetos enquanto *aplicações operacionais* utilizam esses objetos para realizar o controle de acesso [2].

De acordo com o serviço CORBA de segurança, quando um objeto é criado ele automaticamente toma-se membro de um ou mais domínios (domínios de políticas) ficando sujeito às políticas de segurança do domínio. Entretanto, a especificação atual [2] não possui procedimentos para gerenciar os membros dos domínios (incluir, remover, etc) e nem mesmo possui procedimentos para gerenciar os objetos de políticas de segurança. Os domínios de política de segurança são constituídos por coleções de referências de objetos que possuem um conjunto de políticas de segurança comuns e são gerenciados por um objeto *gerente de domínio* [3]. Este gerente tem como função [6]: prover mecanismos para a criação e o acesso aos objetos de política no domínio; e também, manter as estruturas do domínio atualizadas. Entretanto, interfaces para adicionar novas políticas (novos objetos de política) aos domínios ou para modificar o *membership* de domínios ainda não estão padronizadas. A idéia inicial para preencher estas lacunas está sendo proposta no serviço de gerência de *membership* de domínios de segurança (*Security Domain Membership Management Service)* [3], onde é proposta a extensão da interface *DomainManager* com funções administrativas que sirvam para definir e remover objetos de políticas em um domínio.

Mesmo com estas interfaces ainda não padronizadas no *CORBAsec*, sentiu-se a necessidade da introdução deste serviço de gerência de objetos de política no esquema de autorização proposto. O serviço desenvolvido - o *PoliCap* - se baseia em documentos *drafts* (propostas iniciais) liberados pela OMG [3] e certamente não estará muito distante das especificações que deverão em breve ser padronizadas. O *PoliCap* (figura 2) é um serviço que oferece operações tanto para funções *administrativas* quanto *operacionais* sobre objetos de políticas, cumprindo o papel de *gerente de domínio para objetos de política (DomainAccessPolicy)* e de *gerente de direitos para os objetos de direitos requeridos (RequiredRights)* do nosso domínio. O esquema de autorização proposto *(JaCoWeb),* inicialmente, foi definido como constituído por um único domínio e o serviço de política atua como *um serviço centralizado de gerenciamento de políticas e de direitos.*

Aplicações administrativas interagem com o *PoliCap* para gerenciar as políticas e direitos requeridos e, *aplicações operacionais* ou objetos COSS, interagem com o serviço de política para obter, *em tempo de ligação,* as políticas e os direitos requeridos necessários aos controles em tempo de execução sobre as invocações de um método. A idéia é que, *em tempo de ligação,* o serviço de política (gerente de domínio) forneça os objetos *DomainAccessPolicy* e *RequiredRights* que atuam sobre uma invocação. A figura 3 descreve a interface do *PoliCap.*

```
module PoliCap {
#include "SecurityLevel2.idl"
        interface DomainAccessPolicyAdmin:CORBA::DomainManager {
        const CORBA::PolicyType SecClientInvocationAccessDiscretionary = 100;
        void set_policy (       in      CORBA::PolicyType policyType,
                                in      CORBA::Policy     policy);
        void delete_policy (    in      CORBA::PolicyType policyType);
        CORBA::Policy get_local_domain_policy (     in      CORBA::PolicyType   policyType,
                                                    in      Security::SecAttribute  priv_attr,
                                                    in      Security::DelegationState del_state); };
        interface RequiredRightsAdmin:SecurityLevel2::RequiredRights {
                SecurityLevel2::RequiredRights get_local_required_rights (
                        in      CORBA::Identifier target_interface_name); };
};
```

Figura 3. Interface IDL do *PoliCap.*

A operação *set_policy* da *interface DomainAccessPoíicyAdmin* associa a política de

objetos de política *DomainAccessPolicy* e de direitos requeridos *RequiredRights* que são usados localmente na validação de pedidos de acesso a objetos de aplicação. A partir dessa verificação de alto nível são geradas *capabilities* (competências) que são validadas localmente nos servidores remotos, completando então o segundo nível de controle de acesso definido em nosso esquema.

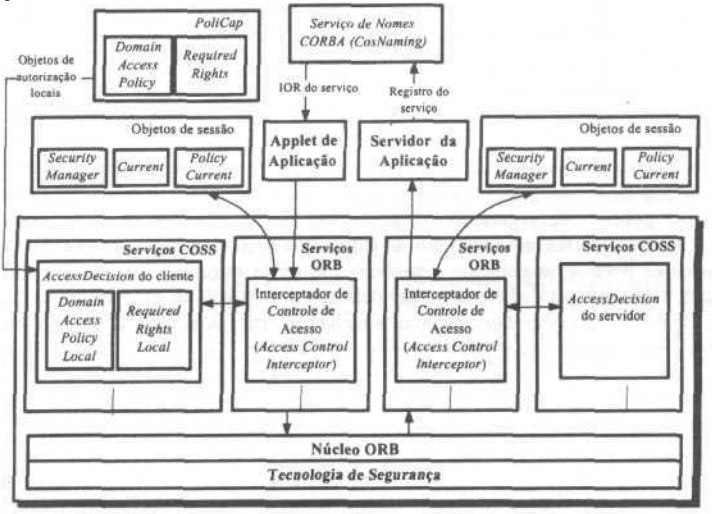

Figura *2.* Estrutura do Controle de Acesso do Esquema de Autorização *JaCoWeb.*

Para montar esses dois níveis de controle de acesso, utilizamos os dois níveis de interceptação definidos no modelo CORBA de segurança e seus objetos de serviço (COSS). A interceptação de alto nível *(Controle de Acesso),* no lado do cliente (*applet* de aplicação), desvia para o objeto de serviço *AccessDecision* que obtém, *em tempo de ligação,* a partir do serviço de política *PoliCap* (detalhado na seção 3.2), os objetos locais *(DomainAccessPolicy* e *RequiredRights)* a serem usados nas verificações do mecanismo de *capabilities em tempo de decisão de acesso.* No lado do servidor da aplicação, a interceptação de alto nível é usada para validar a *capability* recebida com a requisição da invocação.

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 é detalhada na seção 3.3. Além dos controles citados acima, os controles criptográficos também são necessários no esquema e são definidos na forma de objetos de seviço CORBA que usam a tecnologia de segurança residente sob o ORB (objetos *Vault* e *SecurityContext*). Os objetos de serviço *Vault* e *SecurityContext* e o interceptador de chamada segura relativos a associação segura, não estão representados na figura 2 pois são iguais aos da figura 1, seguindo o modelo CORBAsec.

3.2 *PoliCap*

O serviço de política *PoliCap,* proposto neste artigo, tem como objetivo possibilitar o gerenciamento centralizado de objetos de política em um domínio de uma aplicação de objetos distribuídos, suprindo a carência nas especificações do CORBAsec quanto ao gerenciamento de objetos de política. Segundo estas especificações, as políticas de segurança são feitas

de Invocação (*QOPPolicy*, *InvocationCredentialPolicy*, *SecurityMechanismPolicy*, *EstablishTrustPolicy* e *DelegationDirectivePolicy*). Deve então com estas informações decidir quais mecanismos de segurança (cifragem, assinatura, etc.) serão usados na associação segura entre cliente e servidor, estabelecer a confiança entre ambas as partes, garantir a delegação e selecionar a credencial a ser usada na invocação. Esta credencial deve ser compatível com os componentes de segurança que ficam registradas na *IOR* (*InteroperableObject Reference*) do objeto destino. O interceptador de chamada segura também estabelece o contexto de segurança (objeto *SecurityContext)* que será usado para a proteção das mensagens (algoritmos de cifragem, etc).

Em *access decision time,* o *interceptador de controle de acesso* invoca a operação *access_allowed* do objeto *AccessDecision.* A operação *access_allowed é* responsável por autorizar ou não a invocação, obtendo os direitos fornecidos pela política de autorização *DomainAccessPolicy* (*GrantedRights*) e comparando os mesmos com os direitos requeridos (obtidos do objeto *RequiredRights*) para executar a operação na invocação. Em *message protection time,* o *interceptador de chamada segura* no lado do cliente, executa o método *send_message* que chama o método *SecurityContext::protect_message* para proteger as mensagens. Do lado do servidor, o método *receive_message* do interceptador chama o método *SecurityContext::reclaim_message*, para recuperar a mensagem.

3 *PoliCap* - UM SERVIÇO DE POLÍTICA PARA O CORBAsec

O *PoliCap é* um serviço de políticas para objetos distribuídos cujas invocações são reguladas segundo o modelo *CORBAsec.* O *PoliCap* foi projetado no contexto do projeto *JaCoWeb* e corresponde a um primeiro nível de verificação do controle de acesso no esquema de autorização. O segundo nível de controle de acesso corresponde a um mecanismo de *capabilities.* A seguir é descrito sucintamente o esquema de autorização *JaCoWeb.* Na seqüência o serviço de política proposto é apresentado justificando-se a sua necessidade e relevância. Por fim é apresentado o mecanismo de *capabilities.*

3.1 *JaCoWeb*

O *JaCoWeb* 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 [4].

Neste ambiente, aplicações distribuídas são expressas na forma de clientes representados por *appleís* Java, e de objetos servidores de aplicação, visíveis via CORBA. O esquema de autorização define dois níveis de controle de segurança: o *nível global* e o *nível local.* Esses dois níveis são concretizados em objetos de serviço *COSS* e por núcleos de segurança e TCB's (*Trusted Computing Bases),* respectivamente. Os objetos de serviço, descritos na seção 2.1, concentram funções de identificação e autenticação de usuários e os controles de autorização no acesso de objetos de aplicação 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 do nosso esquema.

O *applet* de aplicação (figura 2), depois de autenticado, interage com o serviço de nomes CORBA *(CosNaming)* [5], para obter, a partir do nome do objeto, a referência ou IOR 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, a verificação ocorre *em tempo de ligação,* e uma vez validada a requisição, o serviço de política *PoliCap* fornece versões dos

[18]. O objeto *SecurityManager* mantém informações de estado associadas com a instância do ORB ou com o processo no qual o ORB reside. Por exemplo, referências aos objetos *RequiredRights* e *AccessDecision*, que são utilizadas durante uma invocação, são armazenadas no objeto *SecurityManager*. Informações de estado associadas aos objetos de política do contexto de execução corrente residem no objeto *PolicyCurrent*. Importantes no estabelecimento de uma associação segura entre objetos clientes e servidores, os *objetos de Políticas de Invocação* definem características como a qualidade de proteção das mensagens, as credenciais a serem usadas na invocação, etc. Objetos de *política da invocação* que determinam os tipos de associações possíveis e o objeto *DomainAccessPolicy* que atua durante uma invocação, são acessados a partir do *PolicyCurrent*. O objeto *Current* armazena, no lado do servidor, as credenciais que são enviadas pelo cliente durante o estabelecimento de uma associação segura.

Os objetos *Vault* e *SecurityContext* participam do estabelecimento de uma associação segura. Uma *associação segura* é definida entre objetos cliente e servidor quando é estabelecida a confiança entre as entidades através da autenticação. Esta autenticação se concretiza no instante em que as credenciais do cliente tomam-se disponíveis no lado do servidor e quando é estabelecido o contexto de segurança que será usado para proteger as mensagens em trânsito. O objeto *Vault* cria credenciais em favor do objeto *PrincipalAuthenticator*, de acordo com a tecnologia de segurança subjacente, usada na autenticação. Além disso, o objeto *Vault* também é responsável por criar os objetos de contexto de segurança *SecurityContext* nos lados do cliente e servidor em uma associação. O objeto *SecurityContext* armazena informações do contexto de segurança que é usado para proteger mensagens fornecendo integridade e/ou confidencialidade.

2.2 Interceptadores (*Interceptors*)

Nas especificações do modelo de segurança CORBA, os chamados *serviços ORB* são implementados por *intercepíors*. Logicamente, um *interceptar* 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 de uma invocação de método, ativando um objeto de serviço associado.

No modelo CORBA de segurança são definidos dois *interceptors* que atuam durante uma invocação de método, tanto no cliente como no objeto servidor da aplicação (figura 1): 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 de mensagens correspondentes à invocação.

Estes interceptadores definidos no *CORBAsec* são criados durante o processo de *binding* (ligação) entre dois objetos de aplicação que devem se comunicar. Os dois interceptadores citados estão ligados a diferentes funcionalidades em momentos distintos de uma invocação de método. Em *bind time (em tempo de ligação),* o *interceptador de controle de acesso é* responsável por criar o objeto *AccessDecision* atualizando sua referência no objeto *SecurityManager*. O objeto *AccessDecision*, por sua vez, em *tempo de ligação, é* responsável por disponibilizar os objetos de política *(DomainAccessPolicy)* do domínio. Isto ocorre como conseqüência da execução da operação *get_domain_policy* do gerente de domínio - que resulta na inserção da referência do *DomainAccessPolicy* no objeto *PolicyCurrent*. O *AccessDecision* também localiza ainda o objeto *RequiredRights* do ambiente inserindo a referência ao mesmo no objeto *SecurityManager*.

O *interceptador de chamada segura em tempo de ligação* deve obter e/ou definir as características necessárias para a invocação no lado do cliente, a partir dos *objetos de Políticas*

direitos na execução de operações em todos os objetos de um domínio. A representação de uma política de acesso particular é definida na tabela 1. Para simplificar a administração, o *DomainAccessPolicy* agrega principais usando seus atributos de privilégio. Alguns tipos de agrupamento são *group* (grupo) e *role* (papéis desempenhados pelos principais no sistema). Existe a possibilidade de permitir direitos diferentes a um sujeito que inicia uma invocação (*initiator*) ou que está participando de uma cadeia de delegação de tarefas (*delegate*). Dessa forma, o sujeito no sentido tradicional de uma matriz de acesso é a combinação de um atributo de privilégio e de um estado de delegação. Para lidar com diversas necessidades de expressar a autorização, os direitos são classificados em conjuntos de *tipos de direitos* chamados *famílias de direitos*. O *CORBAsec* fornece apenas a família *Corba* que contém quatro tipos de direitos: g (*get*), *s (sei), m* (*manage*) e *u (use),* embora permita a livre definição de outras famílias de direitos.

Atributo de Privilégio	Estado de Delegação	Direitos Fornecidos (*GrantedRights*)
role:gerente_banco	Initiator	corba: gs-
role:gerente_banco	Delegaté	corba:g---
role:caixa	Initiator	corba: g--u

Tabela 1. Objeto *DomainAccessPolicy.*

Uma política de acesso fornece então direitos aos atributos de privilégio conforme o seu estado de delegação. Em contraste, um objeto *RequiredRights* define que para a invocação de cada operação na interface de um objeto seguro, alguns direitos são necessários ou requeridos *(atributos de controle).* Um exemplo de objeto *RequiredRights* é mostrado na tabela 2. Essa tabela define os direitos requeridos para se ter acesso a cada operação específica de um objeto. Também existe o combinador de direitos que possibilita especificar se um principal necessita todos (*All*) os direitos requeridos para executar uma operação, ou, se é suficiente a posse de qualquer um *(Any)* dos direitos requeridos.

Direitos Requeridos (*RequiredRights*)	Combinador de Direitos	Operação	Interface
Corba:g	AH	Ver_Satdo	Renda_Fixa
Corba:gs	Any	Depositar	Renda_Fixa
Corba:g--u	All	Depositar	Conta corrente

Tabela 2. Objeto *RequiredRights* para as interfaces Renda_Fixa e Conta_corrente.

Os direitos requeridos são atribuídos para *interfaces* e não para *instâncias,* ou seja, todas as instâncias de uma interface terão sempre o mesmo conjunto de direitos requeridos.

Todas as decisões de acesso na invocação dos objetos é feita através de uma interface de um objeto de serviço *AccessDecision* que determina se uma operação a ser executada por um objeto destino específico é permitida ou não. As decisões de acesso dependem dos atributos de privilégio e de controle fornecidos respectivamente, pelo objetos *DomainAccessPolicy* e *RequiredRights.* A regra geral na decisão de acesso pode ser enunciada como segue: um cliente recebe autoridade para executar a operação *m* em um objeto servidor *o,* se a credencial associada com a invocação do método contém um atributo de privilégio que forneça um direito em um domínio de segurança que contém o objeto *o,* e, se o mesmo direito está presente na tabela *"RequiredRights"* da operação *m.* Por exemplo, a política definida na tabela 1 fornece a um principal *caixa,* no estado de delegação *Initiator,* todos (*All*) os direitos requeridos - *g* e *u* - para executar a operação *Depositar* da interface *Conta_corrente.*

Objetos cruciais para o entendimento da dinâmica do funcionamento do CORBAsec são os chamados *objetos de sessão.* Todos os objetos de serviço (COSS) de segurança são acessados através de objetos de sessão *SecurityManager, PolicyCurrent* e *Current* (figura 1), que armazenam informações sobre o contexto corrente de segurança relativo ao processo *(capsule specific)* ou ao fluxo de execução da invocação *(thread specific)* respectivamente

2.1 Objetos de serviço no *CORBAsec*

PrincipalAuthenticator, *Credential*, *DomainAccessPolicy*, *RequiredRights*, *AccessDecision*, *SecurityManager*, *PolicyCurrent*, *Current*, *Vault* e *SecurityContext*, introduzidos nas especificações CORBA, constituem os *objetos de serviço* COSS que implementam os controles de segurança em uma invocação de método.

O objeto *PrincipalAuthenticator* implementa o serviço de autenticação de principais[2] no CORBA. A autenticação define um conjunto de trocas entre o principal e o objeto de serviço *PrincipalAuthenticator* que tem como objetivo final a aquisição de *credenciais* pelo principal. Uma credencial (objeto *Credential)* contém os privilégios de um principal necessários para que este possa acessar os objetos do sistema durante uma sessão. Depois de adquirir credenciais, o principal e os objetos que atuam em seu nome, como clientes, podem iniciar chamadas a objetos servidores.

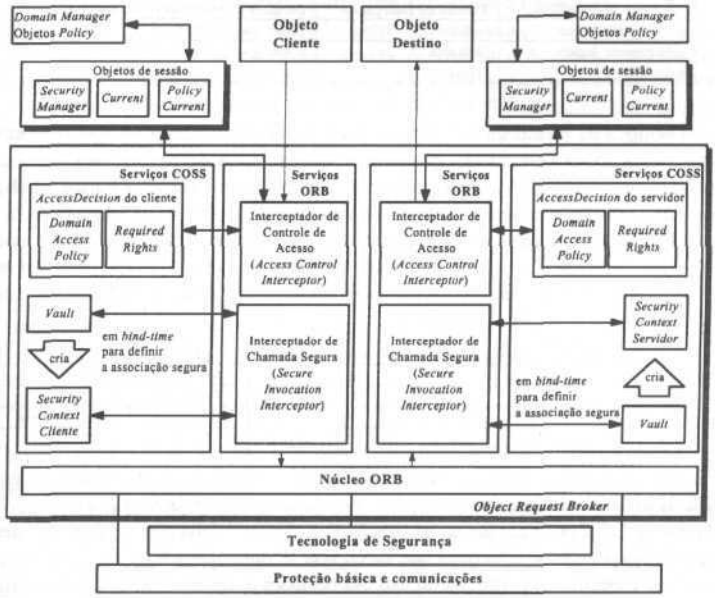

Figura 1. Modelo CORBA de Segurança.

Os controles de autorização enfatizados no modelo permitem que políticas de segurança definidas sejam verificadas em dois níveis: no nível de *middleware* usando objetos de serviço durante uma invocação de método e portanto, de forma transparente para a aplicação; e, no nível da aplicação, que fica consciente dos serviços de segurança do ambiente. Neste trabalho nós nos restrigimos aos controles de autorização transparentes.

No *CORBAsec*, as políticas de segurança são descritas na forma de *atributos de segurança* dos recursos do sistema *(atributos de controle)* e dos principais *(atributos de privilégio).* O objeto *DomainAccessPolicy* representa a interface de acesso à política de autorização discricionária, representando para um conjunto de principais um conjunto de

[2] O *principal* é a entidade (usuário ou processo) responsável e autorizada pela política de segurança a acessar informações ou recursos mantidos pelo sistema.

O serviço de política *PoliCap*, proposto neste artigo, tem como objetivo possibilitar o gerenciamento centralizado de objetos de política em um domínio de aplicações de objetos distribuídos. As nossas propostas suprem as carências identificadas no *CORBAsec* quanto ao gerenciamento de objetos de política e foram desenvolvidas no sentido de atuarem no esquema de autorização do modelo do projeto *JaCoWeb* [4].

O esquema de autorização *JaCoWeb* (http://www.lcmi.ufsc.br/jacoweb) atualmente em desenvolvimento em nossos laboratórios, corresponde a uma estrutura de controle de acesso baseada em dois níveis de controle: um global e outro local, nas estações dos objetos de aplicação. O *PoliCap* constitui o primeiro nível de verificação e atua em tempo de ligação. Um segundo nível de controle de acesso baseado em mecanismo de *capabilities é* realizado em tempo de execução da aplicação, refletindo as políticas do *PoliCap*, permitindo verificações em ambos os lados - no cliente e no servidor - nas invocações de método. O mecanismo de *capabilities é* montado usando as abstrações do próprio *CORBA*. Este esquema de autorização é fundamentado nas especificações do *CORBAsec*, integrando também aspectos dos modelos de segurança Java e Web, na composição da segurança de objetos distribuídos em redes de larga escala.

O artigo no seu plano, inicialmente, apresenta o modelo de segurança do padrão CORBA na seção 2. Na seção 3 é descrita a proposição do serviço de política e o esquema de autorização considerado. Resultados de implementação são mostrados na seção 4 e a avaliação do protótipo desenvolvido é feita na seção 5 de acordo com o Critério Comum de Segurança. A seção 6 apresenta alguns trabalhos correlatos.

2 *CORBAsec*- MODELO DE SEGURANÇA DO CORBA

As especificações CORBA/OMG 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, objetos de serviço (COSS - *Common Object Services Specification*)que suportam serviços comuns, independentes de domínios de aplicação e ainda, em facilidades comuns. Os objetos se comunicam nessa arquitetura via ORB *(Object Request Broker)*. O ORB, num sentido mais genérico, é um canal de comunicação para objetos distribuídos.

A OMG redigiu um documento definindo as principais linhas no que se refere a segurança de objetos distribuídos [2]. O modelo de segurança descrito neste documento estabelece vários procedimentos que tratam da autenticação e da autorização na invocação de um método remoto, da segurança na comunicação entre 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 numa estratificação de sistema (figura 1): o *nível de aplicação* com os objetos de aplicação; o *nível de middleware* formado por objetos de serviço, serviços ORB e o núcleo do ORB *(nível middleware* CORBA); o *nível de tecnologia de segurança* formado pelos serviços de segurança subjacentes; e por fim, *o nível de proteção básica* composto por uma combinação de funcionalidades de sistemas operacionais e do *hardware*. A figura l ilustra os níveis e os componentes principais do CORBAsec.

Nesta figura, os objetos cliente e destino representam o nível das aplicações. Os serviços ORB e objetos de serviço COSS são construídos sobre o núcleo ORB e estendem as funções básicas com qualidades ou controles adicionais, facilitando a implementação de objetos distribuídos. Um conjunto de serviços ORB e serviços COSS é usado no nível de *middleware*, na implementação da segurança no modelo CORBA. A tecnologia de segurança subjacente define algoritmos e protocolos usados na implementação de alguns objetos de serviço do *CORBAsec*.

Nesta versão do protótipo implementamos apenas políticas discricionárias aplicadas no lado do servidor de aplicação. Políticas não-discricionárias ou obrigatórias são objetivos futuros em nosso protótipo. Usando os objetos *Credential* e *Policy* do esquema acreditamos que possamos implementar uma versão do modelo Bell e Lapadula [20] ou ainda de modelos baseados em papéis [21].

Esse protótipo, com os testes realizados, forneceu subsídios para avaliação com relação ao Critério Comum de segurança (ISO 15408), e concluímos que atende o nível EAL3.

REFERÊNCIAS

[1] Bob Blakley, "The Emperor's Old Armor," In *Proc. of the 1996 ACM New Security Paradigms Workshop*, **1996, pp. 2-16, ACM.**

[2] OMG, "Security Service:v1.2 Final," *OMG Document Number 98-01-02,* Nov. 1998.

[3] OMG, "Security Domain Membership Management Service," *Doc. orbos/99-07-21*, Aug. 1999.

[4] Carla M. Westphall and Joni S. Fraga, "A Large-scale System Authorization Scheme Proposal Integrating Java, CORBA and Web Security Models and a Discretionary Prototype," *IEEE LANOMS'99*, Dec. 03-05, pp. 14-25, Rio de Janeiro - Brazil, 1999.

[5] Gerald Brose, "JacORB - A free Java ORB," *Freie Universität Berlin*, Institut für Informatik, Berlin, 1999 (http://www.inf.fu-berlin.de/~brose/jacorb/).

[6] OMG, "ORB Interface," OMG *Document 99-07-08*, June 1999.

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

[8] Li Gong, "A Secure Identity-Based **Capability Systems," *In Proc of the 1989 IEEE Symposium on Security and Privacy*,** pp. 56-63, Oakland, Califórnia, May 1989.

[9] W. Stallings, "Cryptography and Network Security:Principles and Practice," *Prentice Hall*, 2nd edition, July 1998.

[10] Martin Abadi and Roger Needham, "Prudent Engineering Practice for Cryptographic Protocols," *IEEE Transactions on Software Engineering,* Vol. 22, Number 1, pp. 6-15, 1996.

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

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

[13] Graz - University of Technology, "iSaSiLk Toolkit," *Inst. for Applied Information Processing and Communications, Graz* University of Technology, 1999 (http://jcewww.iaik.at/iSaSiLk/isasilk.htm).

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

[15] Murray G. Donaldson et ali, "Guide for the Production of PPs and STs, Version 0.8," *Project ISO/IECJTÇ 1.27.22,* Working Draft, July 1999.

[16] Information Security Systems Organization, "Controlled Access Protection Profile," National Security Agency, Oct. 1999 (http://www.radium.ncsc.mil/tpep/library/protection_profiles/index.html).

[17] Campbell, Roy H. and Qian Tin, "Dynamic Agent-Based Security Architecture for Mobile Computers," Proc. *of the Second* PDCN '98, Austrália, December 1998.

[18] Adiron Inc., "Control - Access Control for ORBAsec SL2 Version 1.0 Alpha," *Adiron LLC CASE Center,* Syracuse University, NY, December 1999 (http://www.adiron.com/).

[19] M. S. Wangham, "Estudo e Implementação de um Esquema de Autorização Discricionária baseado na Especificação CORBAsec," Dissertação de Mestrado, Curso de Pós-Graduação em Engenharia Elétrica, UFSC, Março 2000.

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

[21] K. Beznosov and Y. Deng, "A Framework for Implementing Role-based Access Control Using CORBA Security Service," In *Proc. of 4rd ACM Workshop on Role-Based Access,* Oct. 1999.