UNIVERSIDADE FEDERAL DE SANTA CATARINA

CURSO DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

Estudo e Implementação de um Esquema de Autorização Discricionária baseado na Especificação CORBAsec

Dissertação submetida à
Universidade Federal de Santa Catarina
como parte dos requisitos para a
obtenção do grau de Mestre em Engenharia Elétrica.

Michelle Silva Wangham

Florianópolis, março de 2000.

"Estudo e Implementação de um Esquema de Autorização Discricionária baseado na Especificação CORBAsec"

Michelle Silva Wangham

'Esta Dissertação foi julgada adequada para obtenção do Título de Mestre em Engenharia Elétrica, Área de Concentração em Controle e Microinformática, e aprovada em sua forma final pelo Programa de Pós-Graduação em Engenharia Elétrica da Universidade Federal de Santa Catarina.'

Prof. Joni da Silva Fraga, Dr.
Orientador

Prof. Ildemar Cassana Decker, D.Sc.
Coordenador do Programa de Pós-Graduação em Engenharia Elétrica

Banca Examinadora:

Prof. Joni da Silva Fraga, Dr.
Presidente

Prof. Carlos Becker Westphall, Dr.

Prof. Jean-Marie Farines, Dr.

Prof. Rômulo Silva de Oliveira, Dr.

Filho, desde a tua mocidade aplica-te à disciplina
E até com cabelos brancos encontrarás a sabedoria.
Como o lavrador e o semeador, cultiva-a,
e espera pacientemente seus bons frutos.
Eclesiástico 6, 18-19

Aos meus pais,

Eva e Jura

Agradecimentos

À ti Senhor, que tens guiado os meus passos e me trouxeste em paz até aqui, obrigada. *Por abrigo tenho o teu amor... És meu companheiro, meu melhor amigo...*

Aos meus maiores incentivadores, meus pais, Evani e Jurandy Wangham, e meus irmãos, Danielle e Paulo, obrigada pelo amor, carinho e ensinamentos, fundamentais para a realização deste sonho.

As dificuldades enfrentadas neste mestrado e a saudade da família e dos amigos se tornaram mais leves e menos sofridas graças ao amor, companheirismo e paciência do meu noivo Luiz Magno. Obrigada por dividir comigo as tristezas e alegrias desta vitória.

Agradeço ao Prof. Joni da Silva Fraga, não só pela orientação deste trabalho mas também pela atenção, confiança e incentivo que me impulsionaram a assumir mais um desafio, o doutoramento.

Aos amigos e parceiros do projeto *JaCoWeb Security*, Carla Westphall, Lau Cheuk Lung, cujas contribuições enriqueceram este trabalho, e aos bolsistas Ricardo Padilha e Luciana Sá de Souza, pelas implementações, o meu sincero agradecimento. Foi gratificante trabalhar em grupo com vocês.

Agradeço à *minha família em Florianópolis*, Liane, Gersina, Ivair, Merlin, Sandra, Maurício, Hallthmann, Rubens e Everton, e a toda a galera do Pará, que contribuíram e ainda contribuem para que a minha estadia seja a melhor possível. Aos meus amigos de Belém, *na estrada tão longe e na alma tão perto*, obrigada por todo carinho e amizade.

Aos professores e amigos de luta do LCMI, em especial ao Alexandre Keller, Antônio Carrilho, Augusto Loureiro, Carlos Brandão, César Torrico, Cristiane Paim, Eduardo Machado da Silva, Fábio Benvenuti, Felipe Beck, Fernando Passold, Karen Farfan, Karina Barbosa, Marcos Vallim, Max Queiroz e Nelkis Medina, obrigada pelas contribuições e por tornarem o nosso ambiente de trabalho mais agradável e divertido.

Finalmente, agradeço aos professores e funcionários do Programa de Pós-Graduação em Engenharia Elétrica, por proporcionar um curso de mestrado de qualidade, e à Capes, pelo apoio financeiro.

Resumo da Dissertação apresentada à UFSC como parte dos requisitos necessários para a obtenção do grau de Mestre em Engenharia Elétrica.

Estudo e Implementação de um Esquema de Autorização Discricionária baseado na Especificação CORBAsec

Michelle Silva Wangham
Março/2000

Orientador: Joni da Silva Fraga
Área de Concentração: Controle e Automação
Palavras-chave: CORBA, sistemas de larga escala, segurança
Número de Páginas: 102

RESUMO: Em sistemas de larga escala, como a Internet, pode-se através do *midleware* de comunicação CORBA melhorar os requisitos de reusabilidade, de portabilidade e de interoperabilidade característicos dos sistemas abertos e necessários, por exemplo, em ambientes de negócios. O serviço de segurança do padrão CORBA, o CORBAsec, por sua vez, acrescenta a estes sistemas funcionalidades visando garantir a autenticidade dos usuários, a confidencialidade e a integridade das informações compartilhadas, e o uso legítimo dos recursos e informações do sistema. Neste sentido, este trabalho visa apresentar e discutir o serviço de segurança do CORBA. Com base nos documentos da OMG tem-se neste trabalho, como objetivo, construir um protótipo no sentido de mostrar a aplicabilidade do modelo CORBA de segurança. Os resultados obtidos comprovam a potencialidade do CORBAsec em fornecer segurança às aplicações críticas comuns em ambientes distribuídos e heterogêneos.

Abstract of Dissertation presented to UFSC as a partial fulfillment of the requirements for the degree of Master in Electrical Engineering.

Estudo e Implementação de um Esquema de Autorização Discricionária baseado na Especificação CORBAsec

Michelle Silva Wangham

March /2000

Advisor: Joni da Silva Fraga
Area of Concentration:
Keywords: CORBA, large-scale systems, security
Number of Pages: 102

ABSTRACT: In large-scale systems, such as the Internet, it is possible to improve reusability, portability and interoperability requirements present in open systems and necessary, for example, in e-business, by using the CORBA communication midleware. The CORBA Security Service (CORBAsec) adds new functionalities to these systems, aiming to guarantee the authenticity of the users, the privacy and the integrity of the shared information, and the legitimate use of system resources and information. The purpose of this work is to present and discuss the CORBA Security Service. Based on the OMG documents, our goal is to implement a prototype, in order to show the applicability of the CORBASec model. The obtained results prove that CORBAsec is capable of providing security to critical applications in heterogeneous distributed environments.

SUMÁRIO

Sumário ____ *viii*

Capítulo 1: Introdução ____ *1*

1.1 Objetivo e Justificativa ____ **2**

1.2 Estrutura da Dissertação ____ **3**

Capítulo 2: Segurança de Sistemas Distribuídos ____ *5*

2.1- Introdução ____ **5**

2.2- Conceitos Fundamentais ____ **6**

2.3- Políticas de Segurança ____ **8**

2.4- Violações de Segurança ____ **11**

2.5- Modelos de Segurança ____ **14**

2.5.1- Modelos Baseados em Matrizes de Acesso ____ 14

2.5.2-Modelos baseados em treliças ____ 16

2.6- Mecanismos de Segurança ____ **18**

2.6.1- Criptografia ____ 18

2.6.2- Mecanismos de Autenticação ____ 18

2.6.3- Mecanismos de Controle de Acesso ____ 19

2.7- Critério Comum para Avaliação da Segurança ____ **20**

2.8- Conclusões do Capítulo ____ **22**

Capítulo 3: Serviço de Segurança do padrão CORBA ____ *24*

3.1- Introdução ____ **24**

3.2- Padrão CORBA (*Common Object Request Broker Architecture*) ____ **25**

3.2.1- Arquitetura OMA ____ 26

3.2.2- Arquitetura CORBA ____ 27

3.2.3- Interoperabilidade ____ **30**

3.3- Introdução ao Serviço de Segurança CORBA – *CORBAsec* ____ **30**

3.4- Modelo CORBA de Segurança ____ **32**

3.4.1- Nível de Aplicação ____ 32

3.4.2- Nível de *middleware* CORBA ____ 33

3.4.3- Tecnologia de Segurança ____ 35

3.4.4- Proteção Básica e comunicações ____ 35

3.5- Autenticação de Principais ____ **36**

3.5.1- Objeto *PrincipalAuthenticator* ____ 38

3.5.2- Objeto Credentials ____ 38

3.5.3- Objeto *Current* ____ 39

3.6- Domínios e Políticas de Segurança 39

3.7 – Estabelecimento do Contexto de Segurança 41

3.7.1- *Bind Time* - no lado do Cliente 42

3.7.2- *Bind Time* - no lado do Servidor 43

3.8- Proteção da Mensagem 44

3.9- Autorização e Controle de Acesso 44

3.10- Auditoria 48

3.11- Não-Repudiação 49

3.12- Gerência dos domínios e das Políticas de Segurança 49

3.13- Considerações gerais sobre a segurança CORBA 50

3.14- Levantamento Bibliográfico de Experiências de ORBs seguros 52

3.15- Conclusões do Capítulo 55

Capítulo 4: Esquema de autorização discricionária baseado no CORBAsec *57*

4.1- Introdução 57

4.2- Refinamento da especificação CORBAsec 58

4.2.1- Dinâmica da Autenticação 60

4.2.2- Dinâmica das Atividades Administrativas 62

4.2.3- Dinâmica do Controle de Acesso 63

4.2.4- Estabelecimento e Uso da Associação Segura 66

4.3- Modelo do Protótipo Implementado 72

4.4- Conclusões do Capítulo 73

Capítulo 5: Implementação *75*

5.1- Introdução 75

5.2- Estrutura Geral do Protótipo 76

5.2.1- O Protocolo SSL e o pacote iSaSiLk 77

5.2.2- O *Applet* Cliente e o pacote JDK 1.2.1 79

5.2.3- JacORB 80

5.3- Modelo Implementado 82

5.3.1- Inicialização do ORB seguro 82

5.3.2- Credenciais e Certificados 82

5.3.3- Controle de Acesso Discricionário 83

5.3.4- Integração ORB + SSL 85

5.4- Apresentação dos Resultados 89

5.4.1- Desempenho da integração ORB+SSL 91

5.4.2- Avaliação do Esquema de Autorização Discricionária 92

5.5- Conclusões do Capítulo 95

Capítulo 6: Conclusão *96*

REFERÊNCIAS BIBLIOGRÁFICAS *98*

Capítulo 1: Introdução

Com a expansão das redes de computadores no mundo dos negócios e com a crescente demanda por serviços cada vez mais eficazes, torna-se importante estar habilitado a manipular informações compartilhadas de diversas fontes. Infelizmente, muitas aplicações usadas hoje não foram projetadas para manipular informações compartilhadas, além disso, estas aplicações podem ter sido escritas em diferentes linguagens de programação, necessitando de diferentes plataformas de *hardware* e *software*. Isto exige que as corporações despendam tempo e esforço para produzir uma custosa solução que tente integrar estas aplicações.

A plataforma CORBA (*Common Object Request Broker Architecture*) pode ajudar a solucionar estes problemas. Esta arquitetura, publicada pela primeira vez em 1990 pela OMG (*Object Management Group*)[1], pode ser usada para integrar aplicações e sistemas, fornecendo a flexibilidade necessária para os ambientes de negócios que mudam dinamicamente. As especificações CORBA visam a padronização de uma arquitetura distribuída e orientada a objetos no sentindo de atender aos requisitos de sistemas abertos, tais como: reusabilidade, portabilidade e interoperabilidade.

Como as corporações estão cada vez mais dependentes de seus sistemas de informação, já que suas atividades exigem que as informações sejam processadas em menos tempo, cresce a necessidade da **segurança da informação**. Uma gestão inteligente das informações, com pessoas, processos, infra-estrutura, aplicações e tecnologia apropriadas, é importante para garantir a segurança dos dados estratégicos, viabilizando o sucesso na integração entre tecnologia e negócio [42]. Complementando, Lang [23] afirma que a proteção da informação, em sistemas distribuídos, é muito difícil e complexa, porque a proteção física de computadores, que contêm as informações de negócios, é dificilmente fornecida. Além disso, como as informações devem ser compartilhadas, tecnicamente,

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

qualquer pessoa pode obter acesso aos sistemas, via rede de computadores (por exemplo, a Internet). É necessário garantir às aplicações críticas, como por exemplo, o comércio eletrônico e o *Home/Internet Banking*, meios seguros para operarem. Para preservar a continuidade de seus negócios, as empresas devem: garantir a integridade dos dados e sistemas; manter a confidencialidade dos dados e dos recursos do sistema; garantir a disponibilidade da informação; bem como fornecer mecanismos de autorização aos recursos oferecidos por estas.

Segundo a 5ª Pesquisa Nacional sobre Segurança da Informação [27], as ameaças e os prejuízos decorrentes da falta de segurança na infra-estrutura de tecnologia da informação vêm crescendo no Brasil. Nos últimos dois anos, 30% das empresas brasileiras sofreram algum tipo de invasão, sendo que 39% dos entrevistados[2] não souberam informar se já foram invadidos. A recente invasão em massa dos *sites* do domínio do governo revela a fragilidade dos sistemas, colocando em risco operações críticas e, principalmente, a credibilidade das instituições.

Para contornar estas ameaças, a OMG, em novembro de 1996, acrescentou um Serviço de Segurança CORBA, o *CORBAsec*, à especificação *CORBAservices*, que adicionou características de segurança para informações e aplicações compartilhadas em ambientes distribuídos. Além de fornecer as características de segurança, relevantes aos sistemas de tecnologia da informação (TI), confidencialidade, integridade e disponibilidade, o *CORBAsec* ainda provê características de segurança específicas introduzidas pela arquitetura CORBA.

1.1 Objetivo e Justificativa

Este trabalho tem por objetivo geral a apresentação e discussão do Serviço de Segurança CORBA, como apresentado em (OMG, 1998). Especificamente, este trabalho pretende:

- analisar o modelo e a arquitetura de segurança propostos pelo *CORBAsec*, procurando verificar a aplicabilidade de suas abstrações na segurança de redes de larga escala;

[2]A Módulo Security Solutions S.A., empresa que realizou esta pesquisa, analisou respostas dos mais importantes segmentos da economia brasileira, incluindo instituições financeiras, grandes indústrias e órgãos públicos.

- verificar a viabilidade e a confiabilidade de um esquema de autorização discricionária baseado nas abstrações propostas na especificação CORBAsec, através de um protótipo que visa fornecer segurança a um sistema *Internet Banking* fictício.

Segurança em sistemas distribuídos de larga escala, em ambientes como a Internet, é uma área de pesquisa que tem despertado bastante interesse na comunidade da ciência da computação. Isto é devido à crescente necessidade em integrar com segurança a infra-estrutura de tecnologia da informação aos negócios corporativos. A plataforma CORBA está se constituindo um padrão de *facto* para programação distribuída na Internet [30]. A inclusão de um serviço de segurança a esta plataforma vem consolidar ainda mais este padrão. Entretanto, verificou-se que ainda são poucas as pesquisas realizadas e em andamento, assim como são poucas as empresas que desenvolvem ORBs (*Object Request Broker*) seguros, conforme o modelo do CORBAsec. Espera-se com este trabalho contribuir para um melhor entendimento deste Serviço de Segurança e comprovar a sua aplicabilidade em redes de computadores de larga escala.

O presente trabalho a ser descrito está perfeitamente integrado à linha de pesquisa de segurança de sistemas distribuídos do Laboratório de Controle e Microinformática (LCMI), pertencente à Universidade Federal de Santa Catarina (UFSC), em especial, ao projeto *JaCoWeb Security* [50], desenvolvido nesse Laboratório.

O projeto *JaCoWeb Security* (http://www.lcmi.ufsc.br/jacoweb) visa desenvolver e implementar um esquema de autorização para aplicações distribuídas combinando os modelos de segurança Java/CORBA/Web.

1.2 Estrutura da Dissertação

Este trabalho está divido em seis capítulos.

No primeiro capítulo foi descrito o contexto geral em que o trabalho está inserido, destacando os objetivos desta dissertação e a justificativa do tema escolhido.

Os principais conceitos relacionados com segurança da informação serão apresentados no capítulo 2. Serão discutidas as características desejáveis em sistemas informáticos, a importância das políticas de segurança, as categorias de violações de segurança, bem como, os critérios adotados hoje para a avaliação da segurança.

O capítulo 3 apresentará de forma breve os conceitos e a arquitetura OMA (*Object Management Architecture*) da especificação CORBA, que fornece um *framework* abstrato deste padrão. De forma mais detalhada, o serviço de segurança do CORBA será descrito com base no documento da OMG. Serão apresentados os objetivos do *CORBAsec*, as suas funcionalidades de segurança, a arquitetura e o modelo de referência, os conceitos de domínios e as políticas do serviço de segurança CORBA. Será feita a descrição de como os processos de autenticação, de estabelecimento de contexto de segurança, de controle de acesso e de proteção de mensagens são realizados na arquitetura de segurança CORBA. Ainda neste capítulo, serão descritas as experiências (proprietárias e pesquisas acadêmicas), usando o *CORBAsec* para fornecer ORBs seguros.

Visando apresentar a aplicabilidade das abstrações do CORBAsec, será feita, no capítulo 4, um refinamento dessas abstrações descrevendo um esquema de autorização discricionária. Para o estabelecimento da associação segura, será empregado, como tecnologia de segurança subjacente, o protocolo SSL (*Secure Socket Layer*). O controle de acesso é fornecido através de uma política de autorização discricionária, onde os direitos de acesso a cada recurso são manipulados livremente pelo responsável do recurso ou da informação (geralmente o proprietário do mesmo). E por fim, um protótipo simplificado do esquema de autorização descrito é introduzido.

O capítulo 5 apresentará a descrição detalhada do protótipo implementado do esquema de autorização discricionário, mostrando as ferramentas utilizadas, discutindo as dificuldades encontradas para cumprir o modelo do *CORBAsec* e a viabilidade do esquema proposto.

No capítulo 6 serão discutidos os objetivos atingidos e a sua relevância no contexto da segurança de redes de computadores de larga escala. Serão apresentadas ainda como sugestão propostas para trabalhos futuros nesta área tão relevante no mundo dos negócios, onde a segurança se torna uma característica imprescindível em todos os tipos de aplicações.

Capítulo 2: Segurança de Sistemas Distribuídos

2.1- Introdução

A importância da segurança nas comunicações foi inicialmente reconhecida em ambientes militares e em ambientes onde a segurança nacional estava em jogo (por exemplo, Companhias Públicas Fornecedoras de Energia Elétrica). A criptográfia[1] foi reconhecida como um fator significativo para ganhar os principais conflitos militares do século, incluindo a Segunda Guerra Mundial [11]. O uso difundido das redes distribuídas gera a necessidade de se desenvolverem mecanismos de segurança em diversos ambientes. Diante desta constatação, Voydock [48] apresenta as seguintes tendências:

- O crescente uso de redes de computadores para fornecer acesso remoto às informações torna os ataques às redes de computadores ainda mais atraentes;
- O crescente uso de informações sensíveis que necessitam de segurança , como por exemplo, transferências bancárias, comércio eletrônico e informações proprietárias sigilosas, torna as redes de computadores alvos tentadores;
- Sistemas de computadores conectados via rede de computadores cooperam de diversas formas para fornecer recursos compartilhados para a comunidade de usuários. Como resultado deste compartilhamento, a segurança da informação de uma dada máquina depende das medidas de segurança empregadas na rede e nas outras máquinas.
- O desenvolvimento de novas tecnologias de redes facilita certos tipos de ataques a sistemas de comunicação. Por exemplo, é fácil para um intruso monitorar a transmissão das redes de satélites e de rádios.

[1] Criptografia é a ciência e arte da escrita secreta e do armazenamento secreto de informações [13].

Este capítulo apresenta alguns conceitos essenciais sobre segurança da informação, em que serão discutidas as características desejáveis em sistemas informáticos, a importância das políticas de segurança, as categorias de violações de segurança e os modelos de segurança formais. Os critérios para avaliação da segurança também são introduzidos neste capítulo.

2.2- Conceitos Fundamentais

O conceito de segurança em um sistema informático é identificado com a capacidade de assegurar a prevenção ao acesso e à manipulação ilegítima da informação, ou ainda, de evitar a interferência indevida na sua operação normal [29].

A necessidade de segurança da informação vem crescendo em todos os ambientes de aplicação. Ford, em [11], resume, em uma tabela, os requisitos típicos para alguns destes ambientes (ver Tabela 2.1).

A recente incorporação de proteção de segurança a redes de sistemas abertos[2] vem se mostrado uma tarefa complexa e gigantesca, já que esta representa o casamento de duas tecnologias: a tecnologia de segurança e o desenvolvimento de protocolos de comunicação. Para fornecer redes de sistemas abertos seguras, é necessário empregar técnicas de segurança em conjunto com protocolos de segurança, sendo o último integrado com protocolos de rede convencionais.

A segurança da informação possui quatro objetivos fundamentais (características desejáveis) [11]:

- **Confidencialidade**: garantir que as informações não serão reveladas a pessoas não autorizadas;
- **Integridade**: garantir a consistência dos dados, em particular, prevenindo a criação não autorizada e a alteração ou destruição dos dados, provocados por faltas intencionais ou acidentais;
- **Disponibilidade**: garantir que usuários legítimos não terão o acesso indevidamente negado a informações e recursos;

[2]Os sistemas abertos surgiram devido à reação de consumidores que buscavam uma independência de *hardware* e *software*. Estes sistemas possibilitam a escolha de fabricantes diferentes para diferentes componentes de um sistema, com a confiança de que estes componentes estarão perfeitamente integrados.

- **Uso legítimo**: garantir que os recursos não serão usados por pessoas não autorizadas ou de forma não autorizada.

Ambientes de aplicação	Requisitos
Bancos	- proteger as transações bancárias contra modificações acidentais ou fraudulentas; - identificar os clientes das transações; - proteger os números de identificação pessoal ou PINs (*Personal Identification Numbers*); - garantir a privacidade dos clientes.
Comércio Eletrônico	- assegurar a fonte e a integridade das transações eletrônicas; - proteger a privacidade das empresas; - fornecer assinaturas eletrônicas às transações.
Governo	- proteger as informações sensíveis[3] contra manipulação e divulgação não autorizada; - fornecer assinatura eletrônica aos documentos governamentais.
Companhias Públicas de Telecomunicações	- restringir o acesso de funções administrativas a indivíduos autorizados; - proteger contra interrupções de serviço; - proteger a privacidade dos consumidores.
Redes Privadas	- proteger a privacidade individual e da corporação; - garantir autenticidade de mensagens.
Todas as Redes	- impedir penetrações externas (invasores)

Tabela 2.1: Requisitos típicos para segurança de redes de computadores

Alguns conceitos fundamentais para segurança da informação, tais como: ameaça, proteção, vulnerabilidade, ataque e risco, são apresentados, a seguir de acordo com [11] [2]:

[3]Informações que não são de segurança nacional mas que exigem segurança por outro motivos. Por exemplo, nos Estados Unidos um ato de 1987 sobre segurança de computadores definiu o conceito de informação sensível como toda a informação onde a perda, o uso indevido ou o acesso não autorizado pode, adversamente, afetar o interesse nacional, a conduta dos programas federais, ou a privacidade dos indivíduos (conforme o Ato de Privacidade) [11].

Uma **ameaça** para um sistema de computadores pode ser definida ou identificada como qualquer circunstância ou evento que forneça algum potencial de violação de segurança, comprometendo a integridade, a confidencialidade, a disponibilidade da informação ou a disponibilidade de recursos.

Proteção (do inglês, *safeguard*) são controles físicos, mecanismos, políticas e procedimentos que protegem as informações e os recursos contra ameaças.

Vulnerabilidade são fraquezas nos meios de proteção, ou uma falta destes.

Ataque é uma ação tomada por um intruso malicioso que envolve a exploração de certas vulnerabilidades, visando violações de segurança. Os ataques são classificados como: **passivos**, os que ameaçam somente a confidencialidade dos dados; e **ativos**, que envolvem a modificação não autorizada e a negação de serviço, afetando a integridade e a disponibilidade das informações.

Risco é uma medida do custo de uma vulnerabilidade que incorpora a probabilidade de um ataque ocorrer com sucesso.

Na prática, a concepção de sistemas em que essas violações são totalmente evitadas é muito difícil. Todo o sistema em que a concepção está baseada no compartilhamento de recursos apresenta possibilidades sutis e não esperadas para a transferência de informação não autorizadas entre duas entidades. Pode-se gastar uma quantidade ilimitada de tempo, dinheiro e esforço para buscar segurança, mas nunca se obtém uma imunidade total de problemas

2.3- Políticas de Segurança

Manter as características desejáveis de segurança em um sistema e também assegurar a maneira como são garantidas as características desejáveis, estão fortemente vinculadas à definição de **política de segurança** do sistema.

"*A política de segurança de um sistema é o conjunto de regras e práticas que determinam a maneira pela qual as informações e os outros recursos são gestionados, protegidos e distribuídos no interior de um sistema específico*" [49, pg. 13].

No mundo dos negócios, os especialistas afirmam a importância da elaboração de uma política de segurança corporativa, formalizando procedimentos para o manuseio adequado das informações estratégicas. O gráfico da Figura 2.1 apresentam os resultados

coletados pela 5ª Pesquisa Nacional sobre Segurança da Informação [27], a respeito da adoção de políticas de segurança nas corporações pesquisadas.

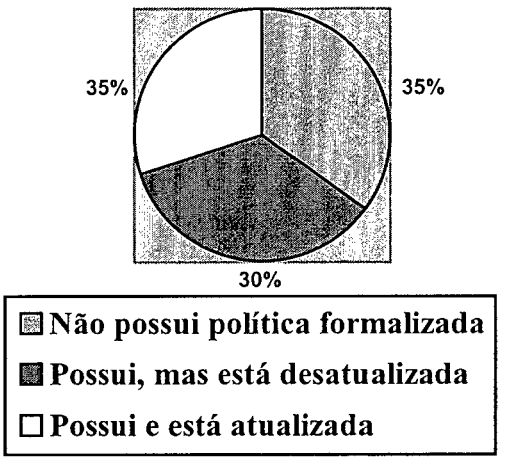


Figura 2.1- Política de Segurança

Observa-se que 65% das corporações apresentam políticas estruturadas, entretanto, quase a metade dos caso, está desatualizada e não integrada ao negócio da empresa. Para 5%, ainda não há previsão para a criação de uma política de segurança.

A noção de política de segurança pode ser destacada em três ramos distintos [29]: as **políticas de segurança física, políticas de segurança administrativa e políticas de segurança lógica**. A primeira se ocupa com tudo que se refere à situação física do sistema a proteger. Neste ramo, em particular, são definidas as medidas contra a violação de segurança por incêndio, catástrofes naturais etc. Deve-se notar que no caso de sistemas distribuídos, a segurança física torna-se cada vez menos eficaz devido ao número significativo de pontos de acesso ao sistema (estações e meios de comunicação). O desenvolvimento da Internet, cada vez com mais pessoas conectadas a partir de seus domicílios graças, a um computador pessoal, é a prova disto.

A política administrativa trata de tudo que resulta da segurança sob o ponto de vista organizacional no seio da empresa, como a seleção do pessoal responsável pela segurança dos sistemas informatizados que fazem parte dessas políticas. Já a política de segurança lógica se interessa pelo conteúdo do sistema de informação. Esta política é encarregada de realizar todos os controles de acesso lógicos, especificando "quem" tem acesso a "que" e em "quais" circunstâncias. A política de segurança lógica pode ser decomposta da seguinte

maneira: cada indivíduo que utiliza um sistema seguro deve se identificar e deve poder provar que ele realmente é a pessoa que pretende acessar um recurso ou informação do sistema. Estas duas idéias são definidas como **política de identificação** e **política de autorização**. Quando um usuário é identificado e autenticado, a política de autorização deve especificar quais são as operações que este usuário particular pode realizar no sistema.

A política de autorização é definida em parte por um conjunto de propriedades de segurança que devem ser satisfeitas e, além disso, por um **esquema de autorização**, que apresenta regras (chamadas regras de transição) que permitem modificar o estado de proteção do sistema. Por exemplo, uma propriedade de segurança poderá ser: "uma informação classificada que não deve ser transmitida a um usuário não habilitado a conhecê-la", enquanto que uma regra do esquema de autorização poderá ser "o proprietário de uma informação que pode conceder um direito de acesso para esta informação a qualquer usuário".

As políticas de autorização, ou mais precisamente, seus esquemas de autorização, se classificam em 2 categorias [29]: as políticas discricionárias e as obrigatórias.

No caso de uma **política discricionária**, os direitos de acesso a cada informação são manipulados livremente pelo responsável da informação (geralmente o proprietário), à sua discrição. O gerenciamento de acesso aos arquivos UNIX constitui um exemplo de mecanismo de controle de acesso baseado nessa política discricionária. Neste caso, três tipos possíveis de acesso são definidos: leitura, escrita e execução. O proprietário de um arquivo pode livremente atribuir ou não estes direitos a si mesmo, a um grupo de usuários, e a outros usuários. Contudo, imagina-se que um usuário U1 confia no usuário U2, mas não confia no usuário U3. U1 dá então direitos de leitura a U2 sob um de seus arquivos, F, mas não a U3. U2 pode então fazer uma cópia de F (pois ele tem o direito de lê-lo) e dar o direito de leitura sob esta cópia a U3. É então impossível, com uma política discricionária, controlar este tipo de vazamento de informação.

Para resolver este tipo de problema, as políticas ditas obrigatórias impõem, para seus esquemas de autorização, regras incontornáveis que se acrescentam às regras discricionárias. Uma **política obrigatória** supõe que os usuários e objetos (recursos do sistema) foram etiquetados. As etiquetas dos objetos seguem uma classificação específica, enquanto que os usuários ou sujeitos do acesso possuem níveis de habilitação. As regras

que regem as autorizações de acesso são baseadas em uma comparação da habilitação do usuário e da classificação do objeto. Estas regras incontornáveis asseguram, por exemplo, que o sistema verifique as propriedades gerais de confidencialidade ou de integridade. Estas regras são freqüentemente utilizadas conjuntamente às regras de uma política discricionária, pois sua capacidade de expressão é relativamente pequena. Neste caso, um usuário será autorizado a manipular uma informação do sistema se o direito de leitura é posicionado sob a informação para ele (controle discricionário) e se ele está habilitado a manipulá-la.

A política obrigatória do **DoD** (*Department of Defense* - Departamento de Defesa dos Estados Unidos), formalizada por Bell e La Padula [4] é um dos exemplos dessas políticas mandatórias. Este, e outros modelos de políticas, são brevemente apresentados na seção 2.5.

Políticas de segurança para proteger sistemas distribuídos devem ser adotadas e projetadas para assegurar níveis de segurança apropriados para as atividades que são executas no sistema; e, mecanismos de segurança devem ser empregados para implementar estas políticas de segurança.

2.4- Violações de Segurança

As violações de segurança podem ser divididas em três categorias distintas [48]:

- **revelação não autorizada da informação**: a informação é divulgada ou revelada para pessoas ou entidades não autorizadas, envolvendo a não verificação da confidencialidade;
- **modificação não autorizada da informação**: a consistência dos dados é comprometida através da criação, alteração, ou destruição dos dados de forma não autorizada (violação da integridade da informação);
- **negação indevida de serviço**: o acesso legítimo da informação ou de recursos é deliberadamente impedido, não garantindo a disponibilidade da informação ou do recurso.

Ford [11] acrescenta ainda uma quarta categoria e constata uma relação direta destas categorias de violação com os objetivos fundamentais da segurança da informação apresentados na seção 2.2:

- **uso ilegítmo**: um recurso é usado por uma pessoa não autorizada ou de forma não autorizada.

O termo *não autorizado*, usado para descrever as categorias de violações, implicam que a revelação, a modificação ou a negação contrariam a política de segurança. O invasor pode ser externo à comunidade de usuários, ou então ser um usuário legítimo da rede. As técnicas de segurança da comunicação são tradicionalmente empregadas para contornar ataques de invasores externos, enquanto que técnicas de autenticação e controle de acesso fornecem a proteção requerida para invasores "internos".

Além dos ataques fundamentais apresentados acima, tem-se o grupo dos ataques primários [11]. A importância deste grupo é significante já que a realização de qualquer um dos ataques deste grupo pode levar diretamente à realização de qualquer um dos ataques fundamentais. Os ataques primários "habilitam" os ataques fundamentais. No grupo dos ataques primários, incluem-se os **ataques de penetração** e os **ataques planejados**. Os principais ataques de penetração são:

- "Mascaramento" (*masquerade*): uma entidade (pessoa ou sistema) se faz passar por outra entidade. Uma entidade não autorizada convence uma proteção (*safeguard*) que ela é uma entidade autorizada, assumindo assim os direitos e privilégios de uma entidade autorizada;
- Transpor controles (do inglês *bypassing controls*): um invasor explora as falhas ou fraquezas de segurança para adquirir direitos e privilégios;
- Violação de autorização: uma pessoa autorizada usa um sistema ou recursos não autorizados (conhecido como ameaça interna).

Os principais ataques planejados são:

- Cavalo de tróia: *software* contendo uma parte invisível ou aparentemente inócua, que, quando executado, compromete a segurança de seu usuário. Um exemplo de cavalo de tróia é uma aplicação que tem aparentemente um propósito legítimo (como a editoração de textos), mas que também tem um propósito fraudulento, por exemplo, copiar documentos de usuários dentro de arquivos ocultos que posteriormente serão lidos pelo invasor que planejou o ataque.

- "Armadilha" (*trapdoor*): uma característica é construída dentro de um sistema ou de um componente do sistema de tal maneira que a entrada de dados específicos permite que políticas de segurança sejam violadas.

Analisando os ataques fundamentais e os primários, têm-se **ataques subjacentes** que também podem habilitar um ataque fundamental. Por exemplo, considerando o ataque fundamental de revelação não autorizada, têm-se os seguintes ataques subjacentes:

- Intromissão (do inglês, *eavesdropping*): a informação é revelada através do monitoramento da comunicação;
- Análise de tráfico: informações são reveladas para entidades não autorizadas através da observação dos meios de comunicação.
- Deslize pessoal: devido a uma negligência uma pessoa autorizada divulga informações para pessoas não autorizadas;
- Lixo informático: informações são obtidas através de meios magnéticos ou impressos que são descartados.

A propagação dos vírus de computador é a principal ameaça à segurança da infra-estrutura de informática. Pragas espalhadas via correio eletrônico como o Melissa e o Chernobil (CIH) atingiram mais de meio milhão de computadores no mundo todo em tempo recorde [27]. Um vírus é um pedaço do código de um programa executável que pode "infectar" outros programas, modificando-os. Um vírus geralmente possui duas funções: infectar outros programas e planejar outros ataques.

No Brasil, além do vírus, outras ameaças se destacam de forma crítica [27]: divulgação indevida de senhas, acessos remotos indevidos, falhas na segurança física, lixo informático e pirataria de *softwares*.

Outros ataques subjacentes estão listadas na Tabela 2.2.

Ataque	Descrição
Repudiação	Uma parte da troca de informações em uma comunicação é falsamente negada.
Mensagem antiga (*replay*)	Uma cópia capturada durante uma comunicação legítima é transmitida novamente com um propósito ilegítimo.
Roubo (*theft*)	Itens de segurança crítica, como por exemplo, cartões de identificação, chaves criptográficas, são roubados.
Esgotamento de recurso	Um recurso (por exemplo, uma porta de comunicação) é deliberadamente usado até que o serviço para outros usuários seja interrompido.

Tabela 2.2: Alguns ataques subjacentes típicos

2.5- Modelos de Segurança

O uso de modelos formais de segurança se torna necessário para organizar a segurança dos computadores e a complexidade que lhes é inerente. Sem uma definição precisa do que a segurança significa e como um computador pode comportar-se, torna-se sem sentido querer saber se um sistema de computador é seguro [22].

Landwehr [22] afirma ainda que é necessário o uso de modelos formais para o projeto de esquemas de autorização. As políticas de autorização são geralmente descritas tendo como base estes modelos que permitem verificar se a política está completa e coerente [29].

Um sistema não só deve ser seguro, como também deve demonstrar isto [22]. Tendo como base um modelo formal e demonstrando que a implementação do esquema de autorização segue este modelo, os projetistas possuem argumentos convincentes de que o sistema é seguro.

2.5.1- Modelos Baseados em Matrizes de Acesso

A noção de matriz de controle de acesso, introduzida por Lampson [21] em 1971, é um modelo simples e geral, baseado nas abstrações dos mecanismos de proteção fornecidos usualmente por sistemas operacionais.

São três os componentes principais encontrados no modelo de matriz de acesso [22]: um conjunto de **objetos** passivos; um conjunto de **sujeitos** ativos, que podem manipular os objetos; e um conjunto de regras que governam a manipulação de objetos pelos sujeitos (**direitos**). Objetos são tipicamente arquivos, terminais, dispositivos e outras entidades implementadas por um sistema operacional. Um sujeito é um processo e um domínio (conjunto de restrições dentre os quais o processo pode acessar certos objetos). Em um dado instante, um sujeito tem o direito de acesso a um objeto, podendo, portanto, executar a operação correspondente sobre o objeto, se esse direito estiver expresso na matriz de acesso.

Na Figura 2.2, tem-se um exemplo do modelo de matriz de acesso. As linhas representam os sujeitos e as colunas os objetos. A interseção de uma linha com uma coluna reflete o modo de acesso entre o sujeito e o objeto correspondente.

Objetos

Sujeitos		**Arquivo A1**	**Procedimento P**	**Segmento S**
	Domínio D1	ler	entrar	ler, escrever
	Domínio D2	ler, escrever	entrar	escrever

Figura 2.2- Exemplo de uma matriz de acesso

A matriz de acesso pode ser vista como a configuração atual do **estado de proteção** do sistema. Certas operações solicitadas pelos sujeitos podem alterar o estado de proteção; por exemplo, se o proprietário de um arquivo o apaga, a coluna correspondente ao arquivo é removida da matriz de acesso. As mudanças de estado de um sistema são realizadas utilizando as regras de transição do modelo. Alguns sistemas concretizam as condições de execução dessas regras em um título de proprietário. Nesse caso, tem-se um modelo discricionário, já que o sujeito que tem o título de propriedade sobre um objeto determina a política de utilização desse objeto, podendo distribuir e revogar os direitos de acesso ao objeto que ele criou [49].

Muitos modelos estão baseados na noção de matriz de controle de acesso, como por exemplo os modelos HRU [15] e Take-Grant [44].

2.5.2-Modelos baseados em treliças

Os modelos baseados em treliças formalizam esquemas de autorização não discricionários. As exigências das políticas de segurança militares ou governamentais do DoD provêm do fato de que se uma informação é conhecida por um inimigo, ela pode se tornar um perigo para toda a segurança nacional. Assim, é importante, em tal abordagem, representar o perigo que pode constituir a divulgação de cada informação do sistema. As informações são classificadas, segundo os quatros níveis de sensibilidade apresentados na Figura 2.3. Além disso, é associado a cada informação um compartimento, constituído de um conjunto de categorias definindo o domínio de informação (tais como "núcleo" ou "criptografia", por exemplo). O nível de segurança de uma informação compreende sua classificação e seu compartimento, isto é, as categorias às quais ela pertence. A cada usuário está associado um nível de segurança representando a confiança que é concedida a este usuário, definido por sua habilitação e um compartimento (a habilitação, assim como a classificação, pode assumir os valores não-classificado, confidencial, secreto, ultra-secreto).

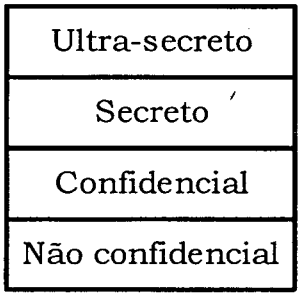


Figura 2.3- Níveis de segurança

- **Modelo Bell e Lapadula [4]**

A política obrigatória do DoD foi formalizada por Bell e Lapadula e compreende um modelo formal multi-nível que permite provar que as regras de controle de acesso que este define satisfazem efetivamente as propriedades de segurança exigidas. Esse tipo de política visa preservar a confidencialidade das informações.

Bell e Lapadula usam máquinas de estado para formalizar seu modelo, definindo o que significa (formalmente) para um determinado estado ser seguro. Consideram ainda quais transições podem ser permitidas de forma que um estado seguro nunca leve a um estado inseguro (coerência).

Além de incluir uma matriz de acesso, que define os direitos de cada sujeito sobre os objetos do sistema, este modelo inclui os níveis de segurança dos sistemas de segurança do DoD [22]: cada sujeito tem um nível de habilitação, chamado de *clearance*, e cada objeto tem uma classificação. Cada sujeito tem também um nível de segurança corrente (varia com a dinâmica do sistema) que não deve exceder o *clearance* do sujeito.

Para um estado ser seguro, duas propriedades devem ser asseguradas [22][2]:

- Propriedade de segurança simples, também conhecida como *no read up*: nenhum sujeito tem direito para ler qualquer informação que tenha uma classificação superior ao *clearance* do sujeito; isto significa que, se um sujeito com um *clearance* **secreto** tentar ler um objeto com a classificação **ultra-secreto,** então este pedido não será permitido.
- Propriedade estrela, também conhecida como *no write down*: nenhum sujeito tem acesso a um objeto cujo nível de segurança não é pelo menos igual ao nível de segurança corrente do sujeito; assim, um sujeito pode ler somente objetos dominados pelo seu nível corrente de segurança e escrever em objetos que dominem seu nível corrente de segurança; isto significa que se um sujeito com *clearance* **ultra-secreto** tenta escrever informações para um objeto **não classificado** do sistema, então isto não será permitido

A propriedade simples tem por objetivo impedir os usuários de ter acesso a informações às quais eles não são autorizados, já a propriedade estrela visa impedir o fluxo de informações de um nível para níveis inferiores de segurança no sistema.

Desde que foi documentado[4], este modelo vem sofrendo extensões e adaptações, assim como vem sendo aplicado em vários projetos de desenvolvimento e implementação de políticas de segurança obrigatórias.

Outros modelos de políticas obrigatórias foram desenvolvidos para a manutenção da integridade. Entre estas políticas, citamos Biba [5] que aplica a integridade a um modelo multi-nível análogo ao de Bell e Lapadula, e Clark e Wilson [8] que formalizam os procedimentos em vigor nos sistemas comerciais [29].

2.6- Mecanismos de Segurança

"*Os mecanismos de segurança em um sistema concretizam a política de autorização definida para o mesmo. Esses mecanismos asseguram que todos os acessos a objetos no sistema são autorizados pela política definida*" [49, pg. 20].

Os mecanismos de segurança para sistemas distribuídos estão baseados no uso de três tipos técnicas [10]: autenticação, controle de acesso e criptografia.

2.6.1- Criptografia

Profissionais de segurança identificaram quatro palavras-chaves que são usadas para descrever as diferentes funções atribuídas ao uso de criptografia em sistemas de informação modernos [13]:

- **Confidencialidade**: a criptografia é usada para "embaralhar" as informações que estão armazenadas em servidores ou que estão expostas em partes do sistemas, como os canais de comunicação, para que invasores não tenham acesso ao conteúdo dos dados;
- **Autenticação**: assinaturas digitais são usadas para identificar o autor da mensagem; pessoas que recebem uma mensagem podem verificar a identidade de quem a assinou;
- **Integridade**: métodos são usados para verificar se as mensagens foram modificadas em trânsito; geralmente isto é feito através de códigos *message digest* assinados digitalmente;
- **Não Repudiação**: recipientes criptografados são criados para que os autores das mensagens não possam negar o envio dessas mensagens.

2.6.2- Mecanismos de Autenticação

O processo de identificação consiste em um conjunto de procedimentos e mecanismos que permitem que agentes externos sejam identificados por algum sistema de computador. Por exemplo, em muitos sistemas operacionais, são fornecidos aos usuários um identificador (*login*) para que estes possam ser reconhecidos pelo sistema. E, o processo de autenticação consiste de um conjunto de procedimentos e mecanismos que permitem a um sistema de computador assegurar que a identidade de um agente externo

esteja correta. Por exemplo, para garantir que a identidade de um usuário está correta, uma senha (*password*) é associada ao *login* do usuário [2].

Mecanismos de autenticação para sistemas distribuídos tomam a forma de um serviço de autenticação. Serviços de autenticação confiam no uso dos controles criptográficos para garantir a segurança. Esses requerem meios seguros para gerar, armazenar e distribuir as chaves criptográficas necessárias em um sistema distribuído – conhecido como **serviço de distribuição de chaves**.

O serviço de autenticação é o mais importante serviço de segurança, já que, até certo ponto, todos os outros serviços dependem deste serviço. A autenticação de entidades contribui de várias maneiras para atingir os objetivos de segurança, tais como:

- fornecer um suporte para mecanismos de controle de acesso, já que a operação depende de um conhecimento seguro das identidades;
- ser um possível meio para fornecer autenticação da origem do dado (quando usada em conjunto com um mecanismo de integridade);
- fornecer identidades seguras associadas às ações de principais para serem gravadas em registros de auditoria (*accountability*);

Existem diferentes mecanismos de autenticação: mecanismos não criptografados, tais como, senha ou PIN's (*Personal Identification Numbers*), *protected passwords*, *one-time passwords* e *challenge response*; e mecanismos criptografados, que estão baseados no princípio de convencer o verificador de que, pelo fato do requerente conhecer uma chave criptográfica, este requerente é quem diz ser.

2.6.3- Mecanismos de Controle de Acesso

O padrão TCSEC (*Trusted Computer System Evaluation Criteria*), usado pelo DoD (conhecido como Livro Laranja), define duas noções essenciais para construção de sistemas seguros: a noção de monitor de referência e de base informática de confiança (TCB – *Trusted Computing Base*).

O **monitor de referência** é a parte do sistema encarregada de validar todos os acessos dos sujeitos aos objetos do sistema. A Figura 2.4 apresenta o monitor de referência na validação dos acessos. O monitor de referência deve obrigatoriamente possuir as três propriedades seguintes [29]:

1- ser inviolável;

2- ser sempre invocado;

3- ser verificável quanto a correção.

A primeira propriedade visa assegurar o isolamento do monitor de referência; não dever ser possível modificar o comportamento do monitor de referência quando o sistema está em funcionamento. A segunda propriedade indica que o monitor deve ser incontornável, ou seja, não deve ser possível a um sujeito acessar um objeto do sistema sem que o acesso não seja controlado pelo monitor de referência. A terceira propriedade determina que o monitor de referência deve ser suficientemente simples para ser analisado e verificado; neste caso, deve-se poder assegurar que o monitor de referência "faz bem" o que ele se propõe a fazer e nada mais. As duas primeiras propriedades estão relacionadas à integridade e à disponibilidade dos monitores e a terceira está relacionada às regras de decisão de acesso (ao modelo formal subjacente).

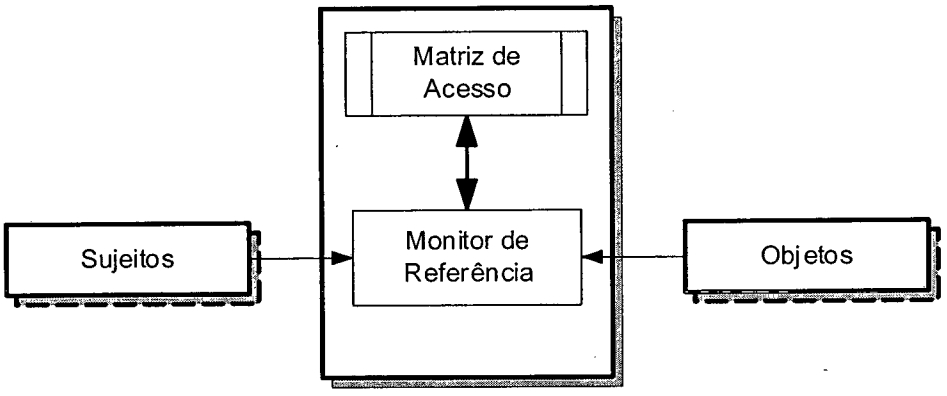


Figura 2.4- Monitor de Referência

Landwehr, em [22], define o **núcleo de segurança** como o conjunto de recursos de hardware e software que permitem realizar um monitor de referência.

A implantação de um esquema de autorização não depende só do controle de acesso. Outros **controles internos**, tais como, serviços de identificação, de autenticação, de não-repudiação e de confidencialidade, também são importantes na concretização de uma política de autorização. Esse controles adicionais formam o que o TCSEC identifica como base informática de confiança ou **TCB**.

2.7- Critério Comum para Avaliação da Segurança

"Em geral, em razão da complexidade de sistemas informáticos e das limitações das técnicas de verificação, a segurança nesses sistemas não pode ser provada ou

estimada. A noção de quanto é seguro um sistema está baseada mais sobre julgamentos que se possa fazer do que sobre medidas exatas." [49, pg. 19]. Essa técnica de verificação baseada em julgamentos é conhecida como uma avaliação qualitativa.

O Critério Comum (*Common Criteria – CC*) para avaliação qualitativa da segurança é o resultado do esforço de várias organizações internacionais para desenvolver um único critério para avaliação da segurança e especificação de sistemas e produtos de tecnologias da informação. Esse critério deriva de padrões anteriores como TCSEC (ou livro laranja que data dos anos 80), ITSEC (critério europeu de 1991), CTCPEC (critério canadense de 1993), FC (união dos critérios europeu e canadense de 1993) e as organizações colaboradoras são: CSE (Canadá), SCSSI (França), BSI (Alemanha), NLNCSA (Holanda), CESG (Inglaterra) e NIST e NSA (Estados Unidos) [9].

O CC está divido em três partes:

- **Parte 1**: apresenta a estrutura e linguagem a ser usada para expressar os requisitos de segurança de um sistema ou de um produto;
- **Parte 2**: apresenta os requisitos funcionais (*functional requirements*) para a segurança de tecnologias de informação de um sistema ou de um produto, definindo o comportamento de segurança desejado;
- **Parte 3**: apresenta os requisitos de garantia (*assurance requirements*) para a segurança de tecnologias de informação de um sistema ou de um produto, que são o fundamento para se conquistar a confiança de que as medidas de segurança solicitadas estão efetivamente e corretamente implementadas.

Conforme o modelo geral do Critério Comum apresentado na Tabela 2.3, cada uma das partes do CC está dirigida para três grupos de pessoas, consumidores, desenvolvedores e avaliadores.

Desde maio de 1998, o grupo de trabalho do CC trabalha em cooperação com a Organização de Padronização Internacional - ISO (*International Standards Organization*) para desenvolver e manter um critério de segurança padrão internacional, baseado no CC. Em dezembro de 1999, o CC tornou-se o padrão ISO 15408.

	Consumidores	Desenvolvedores	Avaliadores
Parte 1: Introdução e Modelo Geral	Para informações básicas e referências.	Para informações básicas e referências no desenvolvimento dos requisitos e formulação das especificações de segurança para os TOEs[4].	Para informações básicas e referências. Guia para construir PPs[5] e STs[6].
Parte 2: Requisitos Funcionais de Segurança	Para guia e referência na formulação da declaração dos requisitos das funções de segurança.	Para referência na interpretação das declarações dos requisitos funcionais e formulação das especificações funcionais dos TOEs.	Declaração obrigatória do critério de avaliação para determinar se o TOE cobre efetivamente as funções de segurança solicitadas.
Parte 3: Requisitos de Garantia de Segurança	Para guiar a determinação dos níveis de garantia requeridos.	Para referência na interpretação das declarações dos requisitos funcionais e determinação das abordagens de garantia dos TOEs.	Declaração obrigatória do critério de avaliação para determinar a garantia do TOE e para avaliar os PPs e STs.

Tabela 2.3 – Modelo Geral do Critério Comum [46].

2.8- Conclusões do Capítulo

Desde que a infra-estrutura da tecnologia da informação, se torna mais extensa e complexa, aumenta o desafio de integrar com segurança a tecnologia com os diversos ambientes de aplicação, como os apresentados na Tabela 2.1. Multiplicam-se os pontos vulneráveis na rede e os ataques chegam de toda a parte. Os riscos de violações de segurança causados por funcionários insatisfeitos, hoje são agravados pela ação de *invasores*.

[4] O objeto de avaliação (TOE – *Target of Evaluation*) é a parte do produto ou sistema que está sujeita à avaliação.

[5] *Profiles* de Proteção (PP - *Protection Profile*) são os conjuntos de requisitos e objetivos que permitem aos consumidores e desenvolvedores criar conjuntos padronizados de requisitos de segurança de acordo com suas necessidades.

[6] Do ingles, *Security Target*, é o alvo de segurança que contém os objetivos e requisitos de segurança de um TOE específico.

Os profissionais de segurança devem estabelecer metas para a construção de um sistema informático seguro. Como cada organização tem diferentes preocupações com a segurança, devem-se, primeiramente, identificar as necessidades de segurança, confidencialidade, integridade, disponibilidade, controle de acesso e auditoria, para um dado sistema.

Questões básicas como: "*o que precisa ser protegido ?*", "*contra o que precisa-se proteger ?*" e "*quanto dinheiro e esforço será gasto para se obter a proteção adequada?*", constituem a base do processo para estabelecer as taxas de risco de um sistema [13]. Se os riscos são conhecidos, pode-se planejar políticas de segurança e implementar mecanismos para reduzi-los.

A implantação de esquemas de autorização ajuda no combate à prevenção de invasores. Aplicações que manipulam informações sensíveis (*Home/Internet Banking* e comércio eletrônico) costumam empregar políticas discricionárias para evitar acessos não autorizados. Para aplicações que exigem uma maior preocupação, como as que envolvem a segurança nacional de um país, sugere-se acrescentar ao esquema anterior políticas obrigatórias para um controle ainda mais rigoroso. Quando baseados em modelos formais de segurança, os esquemas de autorização podem ser mais facilmente verificados quanto a sua completude e coerência.

A partir de uma análise qualitativa, pode-se ainda verificar a segurança de um sistema, tendo como base o Critério Comum para Avaliação da Segurança, recentemente padronizado pela ISO.

Capítulo 3: Serviço de Segurança do padrão CORBA

3.1- Introdução

A necessidade de compartilhar informações de diversas fontes torna as atividades de projeto e implementação de *software* difíceis e custosas. Em [23], tem-se um cenário desta situação:

> Uma grande empresa possui sistemas que têm evoluído durante décadas, incluindo mainframes, minicomputadores, PCs, LANs, WANs, e múltiplos sistemas de administração de banco de dados. Estes sistemas executam diversas aplicações em diferentes sistemas operacionais. Além disto, devido à necessidade de novos negócios e de mudanças organizacionais, há freqüentes mudanças nos softwares. E, a companhia necessita integrar estes sistemas.

O padrão CORBA (*Common Object Request Broker Architecture*) pode ser usado para integrar aplicações distribuídas, aumentando a reusabilidade, a portabilidade e a interoperabilidade de *softwares* orientados a objetos. Esta arquitetura torna mais fácil a programação, permitindo assim criar aplicações heterogêneas sobre as principais plataformas de *hardware* e sistemas operacionais disponíveis. O CORBA tem ainda por objetivo encorajar os desenvolvedores de aplicações abertas, para que essas sejam usadas como componentes dos sistemas de larga escala [3].

Controle de acesso em sistemas de larga escala (por exemplo, milhões de objetos na Internet) é problemático. Como novos componentes estão constantemente sendo adicionados, apagados e modificados, as políticas de segurança mudam dinamicamente. Além disto, nestes sistemas pode haver ainda muitos domínios de política de segurança diferentes, cada um garantindo requisitos de segurança de uma parte do sistema. Devido a estas complexidades, os esquemas de autorização se tornam difíceis de serem coerentes.

Sistemas de objetos distribuídos são formados por uma grande quantidade de camadas de *softwares* a serem implementadas, como por exemplo, a ferramenta de

programação, *middleware*[1], serviços de segurança, tecnologia de segurança, sistema operacional, dentre outras que possam existir. Vulnerabilidades podem ocorrer entre cada uma dessas camadas, pois a interação entre as mesmas é bastante complexa. A complexidade dos vários níveis é ainda mais complicada em sistemas onde a garantia da segurança é amplamente distribuída.

Visando acrescentar ao CORBA um suporte seguro às aplicações distribuídas, a OMG[2] (*Object Management Group*) redigiu um conjunto de relatórios definindo as principais linhas no que se refere à segurança de objetos distribuídos [33]. Muitos objetivos do serviço de segurança CORBA (*CORBAsec*) são relevantes a todos os sistemas de tecnologia da informação usados em aplicações de larga escala. Contudo, alguns dos objetivos de segurança são específicos da arquitetura CORBA.

Este capítulo inicia com uma breve apresentação da arquitetura OMA (*Object Management Architecture*), descrevendo a infra-estrutura conceitual sobre as quais todas as especificações da OMG estão baseadas, e, em seguida, aborda sucintamente os componentes do padrão CORBA.

Com base no documento da OMG, o serviço de segurança CORBA é descrito de forma mais detalhada. São apresentadas as funcionalidades de segurança, a arquitetura e o modelo de referência e os conceitos de domínios e políticas.

Ainda neste capítulo, serão descritos alguns produtos e experiências acadêmicas que desenvolvem ORBs seguros, analisando a conformidade desses ao modelo de segurança CORBA.

3.2- Padrão CORBA (*Common Object Request Broker Architecture*)

Uma das primeiras especificações publicadas pela OMG foi a especificação CORBA[3], que detalha as interfaces e características do componente ORB da OMA. Esta especificação introduz um conjunto de conceitos e mecanismos padronizados que permitem a implementação de objetos distribuídos, cujo modelo de interação adotado é o cliente/servidor.

---

[1] O termo em inglês *midleware* é utilizado para designar camadas de software acima do sistema operacional que ofereçam algum serviço de suporte às aplicações.

[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.

[3] *Common Object Request Broker: Archicture and Specification* [34]

O CORBA é um *midleware* de comunicação porque separa uma aplicação dos detalhes e complexidades do núcleo de comunicação [24]. Esta arquitetura define os objetos como as unidades básicas de distribuição, sendo que cada objeto suporta um conjunto de operações por meio de uma interface. Esta interface é especificada em uma linguagem de definição de interface ou **IDL** (do inglês, *Interface Definition Language*), que é independe da linguagem de implementação dos objetos [34].

3.2.1- Arquitetura OMA

A arquitetura OMA define um *framework* para comunicações entre objetos distribuídos, no qual todas as especificações da OMG estão baseadas. Esta arquitetura é composta de um **Modelo de Objeto** e de um **Modelo de Referência**. O Modelo de Objeto define como os objetos distribuídos podem ser descritos em ambientes heterogêneos, enquanto o Modelo de Referência caracteriza as interações entre esses objetos [47].

No Modelo de Objeto da OMA, um objeto é uma entidade encapsulada com uma identidade distinta cujos serviços podem ser acessados apenas através de interfaces bem definidas.

A Figura 3.1 apresenta os componentes do Modelo de Referência da OMA. São eles:

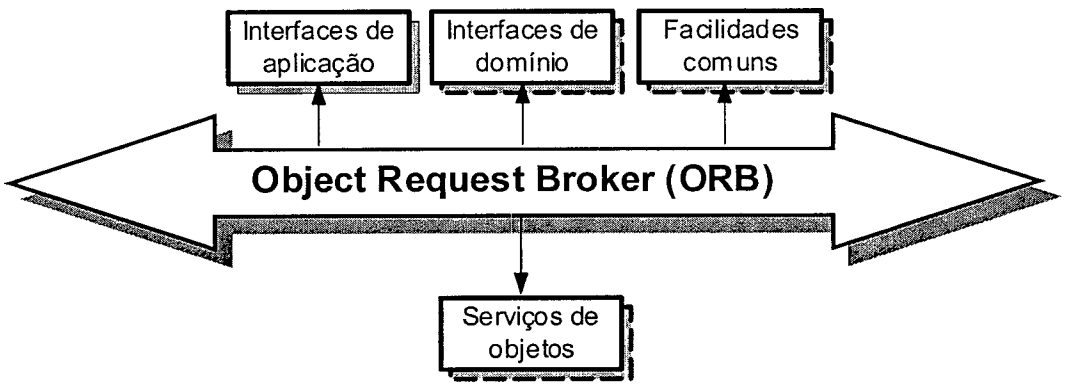


Figura 3.1- Componentes da arquitetura OMA

- ***Object Request Broker* (ORB)** – É o componente central da arquitetura OMA e equivale à arquitetura CORBA propriamente dita. Um ORB é o mecanismo básico no qual os objetos, transparentemente, fazem uma requisição para - e recebem respostas de - um outro objeto, podendo estar na mesma máquina ou em máquinas diferentes. Um objeto cliente não necessita estar ciente dos

mecanismos de comunicação, da implementação do objeto servidor e da localização deste objeto. Constata-se, então, que o ORB é a base para a construção de aplicações com objetos distribuídos e para a interoperabilidade entre as aplicações, em ambientes homogêneos e heterogêneos [24].

- **Serviços de objetos** (COSS – *Common Object Services Specification*) – São serviços (interfaces e objetos), independentes do domínio da aplicação, de propósitos gerais que são fundamentais para o desenvolvimento de aplicações formadas por objetos distribuídos, ou que fornecem uma base universal para a interoperabilidade das aplicações. Em conjunto com o ORB, os serviços de objeto (ou serviços CORBA) estão relacionados a aspectos de infra-estrutura do sistema distribuído. Os serviços CORBA, por exemplo, *Naming Service*[4], *Trading Service*[5] e *Security Service*[6], estão descritos em [35].
- **Facilidades comuns** [CORBAfacilities]– Assim como as interfaces dos serviços de objetos, as facilidades comuns são interfaces que podem ser compartilhadas por várias aplicações, mas que, ao contrário dos serviços de objetos, são orientadas a aplicações mais próximas dos usuários (*end-user applications*).
- **Interfaces de domínio** – Exercem o mesmo papel dos serviços de objetos e facilidades comuns, porém são interfaces orientadas a domínios de aplicação específicos. Por exemplo, domínios como telecomunicações ou finanças.
- **Interfaces de aplicação** – Os objetos de aplicação são desenvolvidos especificamente para uma dada aplicação. Como estas interfaces consistem na aplicação propriamente dita, as interfaces não são padronizadas pela OMG.

3.2.2- Arquitetura CORBA

A estrutura introduzida pelo CORBA é constituída por um núcleo ORB que implementa abstrações e semânticas relacionadas com a comunicação entre objetos

[4] *Naming Service* ou Serviço de Nomes. Antes de usar um objeto CORBA, um programa cliente deve obter um identificador do objeto conhecido como referência de objeto. Este serviço permite a um cliente localizar referências de objetos baseados em nomes abstratos de objetos (referências de objetos interoperáveis ou IORs – *Interoperable Object References*).

[5] *Trading Service*. Permite a um cliente localizar uma referência de objeto baseada nas propriedades desejáveis de um objeto.

[6] Este serviço é detalhadamente descrito no Capítulo 4.

distribuídos. A Figura 3.2 ilustra os componentes da arquitetura ORB do CORBA[7]. A descrição destes componentes estão a seguir:

- **Núcelo ORB** – O núcleo do ORB, em conjunto com outras estruturas, é responsável por encontrar a implementação de objeto adequada, por distribuir requisições de clientes ao objeto destino e retornar a resposta da solicitação aos clientes. Vinosky [47] afirma que a característica chave do ORB é a transparência da comunicação cliente/servidor

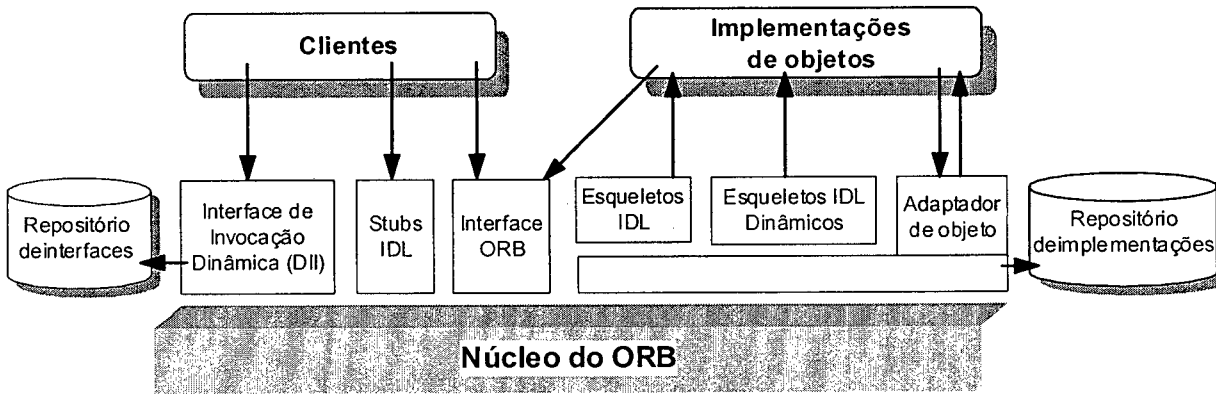


Figura 3.2 – Arquitetura CORBA

- **Implementações de Objetos** – As implementações de objetos definem as operações que implementam as interfaces IDL do CORBA. A IDL da OMG é apenas uma linguagem declarativa, e não é diretamente usada para implementar aplicações distribuídas. A tradução de uma IDL (*language mapping*) é quem determina como as características da IDL serão mapeadas para uma dada linguagem de programação (C, C++, Ada, Cobol, Java, etc.).
- **Clientes** – Os clientes são os objetos que, de forma simples, invocam as operações das implementações de objetos (servidores).
- **Interface ORB** – Para separar as aplicações dos detalhes de implementação, a especificação CORBA definiu uma interface abstrata para o ORB. Esta interface provê várias funções de auxílio, como converter referências de objetos para *strings* e vice-versa [3].

[7] Esta arquitetura segue a especificação do CORBA 2.0.

- ***Stubs* IDL ou interface de invocação estática** – O *stub* IDL é o mecanismo que efetivamente cria e envia as requisições em nome de um cliente até o núcleo do ORB, que é o encarregado de localizar a implementação do objeto adequado e de ativá-la. Os *Stubs* descrevem os serviços fornecidos pelas implementações de objetos.
- **Esqueletos IDL estáticos** – O Esqueleto IDL é um mecanismo que entrega as requisições dos clientes à implementação do objeto CORBA. Em conjunto, os *stubs* e os esqueletos permitem, através do núcleo do ORB, a comunicação entre os clientes e as implementações de objeto.
- **Interface de Invocação Dinâmica ou DII (*Dynamic Invocation Interface*)** – Usando DII um cliente pode invocar requisições diretamente ao objeto destino sem ter conhecimento, em tempo de compilação, das interfaces do objetos. Aplicações usam uma interface de invocação dinâmica para emitir pedidos aos objetos sem requerer uma *stub* IDL específica.
- **Esqueletos IDL dinâmicos ou DSI (*Dynamic Skeleton Interface*)** – Análogo ao DII, os esqueletos IDL dinâmicos permitem ao núcleo ORB entregar requisições a uma implementação de objeto que não tem conhecimento, em tempo de compilação, do tipo do objeto que está implementando.
- **Adaptador de Objeto** – O adaptador de objeto auxilia o núcleo do ORB na entrega de requisições e na ativação de objetos. Associa uma implementação de objeto ao ORB.
- **Repositórios de interfaces** – As interfaces de objetos (IDLs) podem ser adicionadas a um repositório de interfaces. Para usar as interfaces de invocação dinâmica, os clientes necessitam criar uma requisição que deve incluir a referência do objeto, a operação e a lista de parâmetros. A identificação do objeto e da operação pode ser obtida em um repositório de interfaces, que é um banco de dados que fornece armazenamento persistente de definições de interface de objetos.
- **Repositório de implementações** – O repositório de implementações contém informações que permitem ao ORB localizar e ativar implementações de objetos em tempo de execução.

3.2.3- Interoperabilidade

A interoperabilidade resulta na possibilidade das aplicações interagirem utilizando ORBs distintos. O padrão CORBA 2.0 introduz uma sintaxe de transferência geral para interoperabilidade entre ORBs, conhecida como *General Inter-ORB Protocol* (GIOP). A OMG também especificou o protocolo *Internet Inter-ORB Protocol* (IIOP) que usa o TCP/IP como seu protocolo de transporte. O protocolo IIOP é o protocolo para comunicações entre objetos e aplicações que usam a Internet.

3.3- Introdução ao Serviço de Segurança CORBA – *CORBAsec*

O *CORBAsec* é uma especificação aberta para segurança em sistemas de objetos distribuídos. Em maio de 1996, a OMG adotou a primeira especificação de segurança CORBA completa (CORBAsec 1.1). Desde então, o CORBAsec sofreu duas revisões e está atualmente na revisão 1.2 [36].

A arquitetura de segurança CORBA foi projetada para atender os seguintes objetivos gerais de segurança para aplicações de larga escala [36]:

- **Simplicidade** para entender os conceitos e para administrar a implementação;
- **Consistência** em ambientes distribuídos. Por exemplo, suportar políticas de acesso consistentes;
- Fornecer segurança para pequenos sistemas, sistemas locais e sistemas de larga escala, podendo dividir os sistemas em domínios gerenciáveis que podem interoperar. Este objetivo é conhecido como ***scalability***;
- **Usabilidade** para usuários finais, fornecendo segurança de forma transparente. Usabilidade para administradores, garantindo boa flexibilidade. E, usabilidade para implementadores, possibilitando a esses estar ou não cientes da segurança que protegerá sua aplicação;
- **Flexibilidade** em políticas de segurança;
- **Independência da tecnologia de segurança**;
- **Portabilidade** para as aplicações;
- **Interoperabilidade** entre ORBs diferentes, entre sistemas com e sem segurança, entre tecnologias de segurança diferentes, e entre domínios de um sistema distribuído, que pode ainda suportar diferentes políticas de segurança;

- A segurança não deve impor um **desempenho** inaceitável;
- As interfaces de segurança devem ser puramente **orientadas a objeto**.

Além dos objetivos gerais, tem-se os objetivos específicos da arquitetura de segurança CORBA, tais como [24]:

- Fornecer segurança de sistemas heterogêneos com ORBs diferentes;
- Usar encapsulamento para esconder a complexidade de mecanismos de segurança sobre interfaces simples;
- Permitir o polimorfismo nas implementações de objetos baseados em diferentes mecanismos subjacentes;
- Garantir que invocações de objetos serão protegidas quando requeridas pela política de segurança;
- Garantir que o controle de acesso e a auditoria requerida serão executados sobre a invocação do objeto.

Existem diversas medidas para contornar ou diminuir as vulnerabilidades de sistemas de objetos distribuídos descritas na seção 2.6. Como por exemplo, as **funcionalidades de segurança** dos modelos de objetos descritos na especificação do CORBAsec, tais como:

- Identificação e autenticação;
- Autorização e controle de acesso (prevenindo acessos não autorizados);
- Auditoria de segurança;
- Segurança da comunicação (garantir integridade e confidencialidade);
- Não-repudiação;
- Administração da segurança (políticas e domínios de segurança).

O CORBA define ainda vários esquemas para delegação de privilégios. Em um sistema composto por objetos, um objeto cliente pode requerer a execução de uma operação a outro objeto (objeto intermediário), que por sua vez requer que outros objetos realizem essa tarefa. Isso geralmente resulta em uma cadeia de invocações de outros objetos. Um objeto intermediário, além de solicitar a outros objetos uma operação, deve também poder transmitir uma parte dos seus direitos ao objeto destino (servidor da aplicação). Entretanto essa delegação deve ser controlada de forma que o objeto ao qual os

direitos são delegados não possa executar outras operações, além das que as duas partes estão de acordo.

Os ORBs seguros não necessitam suportar todas as funcionalidades de segurança definidas na especificação do CORBAsec. A especificação define dois níveis de funcionalidade em que esses ORBs podem se enquadrar de acordo com os serviços que estes dispõem.

O **nível 1** (*SecurityLevel1*) fornece segurança às invocações entre cliente e objeto destino, com delegação simples de atributos de segurança, para as aplicações que não são cientes da segurança, garantindo controle de acesso aplicado pelo ORB e auditoria de eventos do sistema. Este nível possui ainda uma interface na qual as aplicações, cientes das características de segurança, podem acessar atributos de segurança, que poderão ser aplicados nas políticas de segurança da própria aplicação (por exemplo, controle de acesso). A funcionalidade de não-repudiação é opcional.

O **nível 2** de funcionalidade (*SecurityLevel2*) suporta, além da funcionalidade de nível 1, outras interfaces de aplicação e interfaces de administração. Facilidades para a administração de políticas de segurança são introduzidas neste nível, permitindo que aplicações determinem suas próprias políticas de autorização. Opcionalmente, este nível pode ainda fornecer não-repudiação e interoperabilidade segura.

3.4- Modelo CORBA de Segurança

O modelo estrutural do CORBAsec possui quatro níveis que são usados durante uma invocação [36]: 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 3.3 apresenta os níveis e os componentes principais do modelo CORBAsec. A seguir estão descritos cada um desses níveis e seus respectivos componentes.

3.4.1- Nível de Aplicação

Os objetos cliente e destino da Figura 3.3 representam o nível de aplicação. Muitos componentes de aplicação não são cientes da segurança e confiam no ORB para chamar,

de forma transparente, os serviços de segurança requeridos durante a invocação de objetos. Entretanto, algumas aplicações garantem sua própria segurança ativando diretamente os serviços de segurança.

3.4.2- Nível de *middleware* CORBA

O núcleo do ORB é definido na arquitetura CORBA como "a parte do ORB que fornece a representação básica dos objetos e a comunicação dos pedidos". O núcleo do ORB, portanto, suporta a funcionalidade mínima para habilitar um cliente a invocar uma operação de um objeto destino, com as transparências de distribuição requeridas na arquitetura CORBA. Os serviços ORB e os objetos de serviço COSS de segurança do modelo da Figura 3.3 são construídos sobre o núcleo do ORB e estendem as funções básicas com qualidades ou controles adicionais, facilitando assim a implantação das funcionalidades de segurança.

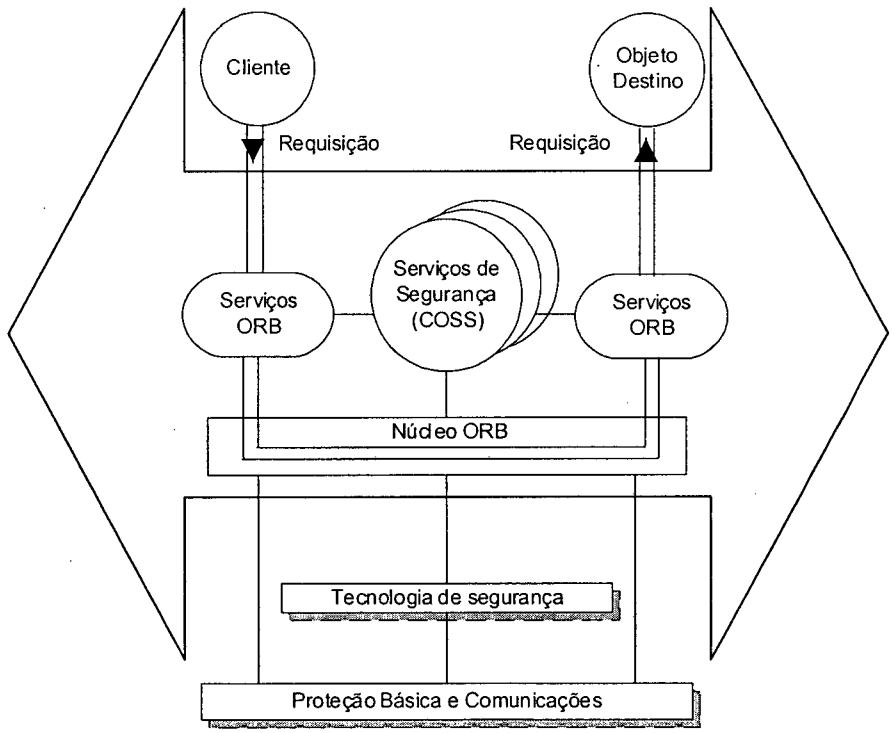


Figura 3.3 – Modelo Estrutural do CORBAsec

Na especificação do modelo *CORBA* de segurança, os chamados serviços ORB são implementados por interceptadores (ou *interceptors*). Logicamente, um interceptador é interposto no caminho de uma chamada entre um cliente e um destino. Cada serviço COSS relacionado com a segurança é associado a um interceptador cuja finalidade é provocar o

desvio transparente ao objeto de serviço correspondente. Os serviços COSS de segurança podem invocar os mecanismos de segurança subjacentes para garantir algumas funcionalidades de segurança, tais como: autenticação, controle de acesso, auditoria, não-repudiação e invocações seguras.

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

A Figura 3.4 ilustra, de forma mais detalhada , o modelo de segurança CORBA, apresentando os serviços ORBs abordados acima e os serviços COSS de segurança que são acionados pelos dois interceptadores.

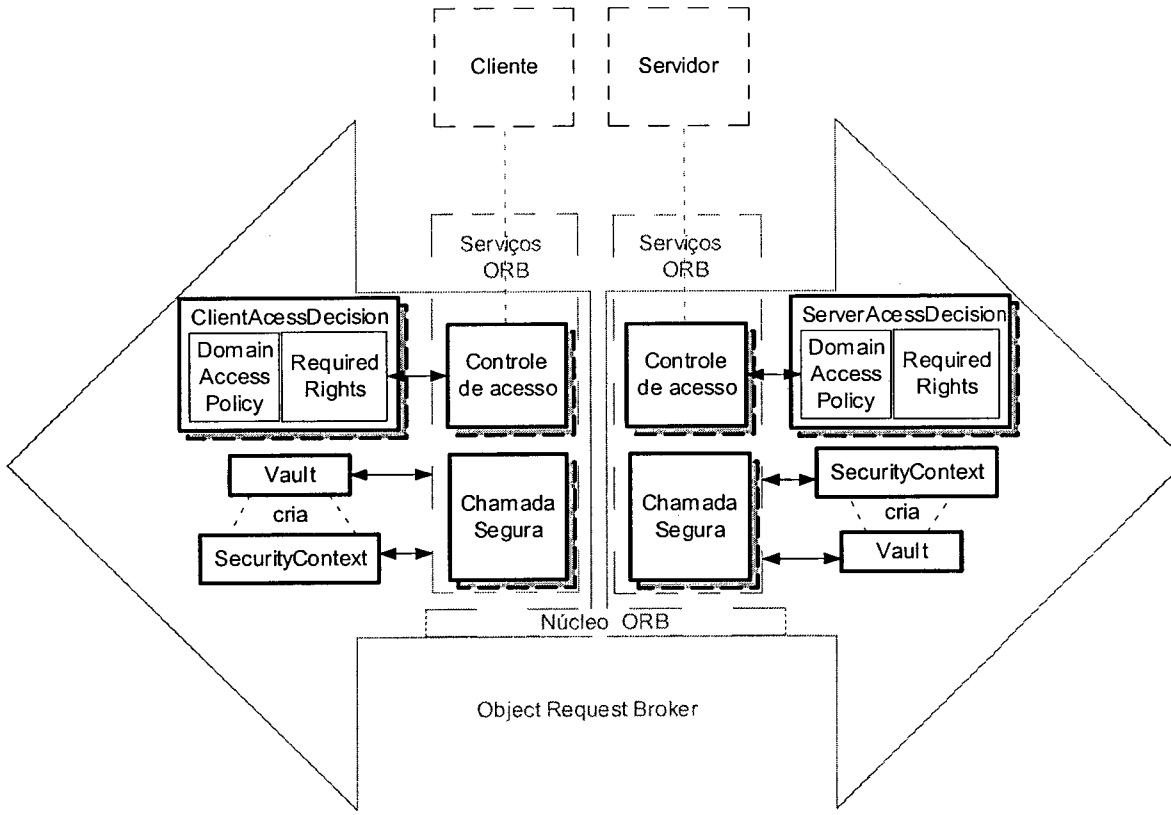


Figura 3.4- Serviços de Segurança do Modelo CORBAsec

Analisando a Figura 3.4, observa-se que no interior das setas estão contidos o interceptador de controle de acesso que está representado pelo objeto **Controle de Acesso**

e o interceptador de chamada segura que está representado pelo objeto **Chamada Segura**. Os objetos de serviço que implementam os controles de segurança nas especificações CORBA são: o ***Principal Authenticator*** que corresponde ao serviço de autenticação de principais no CORBA; o ***Vault*** que estabelece as associações seguras entre os clientes e servidores com os respectivos contextos de segurança; o ***SecurityContext*** que representa o contexto da associação segura , o ***Credential*** que representa as credenciais ou direitos do cliente na sessão; o ***DomainAccessPolicy*** que representa a interface de gerenciamento de políticas de autorização discricionárias e concede a um conjunto de principais, direitos específicos para executar operações sobre todos os objetos do domínio [20]; o ***RequiredRights*** que armazena as informações sobre os direitos (*rights*) necessários para executar cada método de cada interface do sistema; e os objetos ***AccessDecision*** responsáveis por interagir com os objetos ***DomainAccessPolicy*** e ***RequiredRights*** para determinar se uma dada operação sobre um objeto destino específico é permitida. Existem, também, outros objetos de serviço relacionados com a não-repudiação e auditoria.

3.4.3- Tecnologia de Segurança

Como apresentado na Figura 3.3, os objetos de serviço no modelo CORBA de segurança isolam aplicações e o ORB da tecnologia de segurança. Esta última consiste em uma camada subjacente que implementa várias funcionalidades dos objetos de serviço do CORBA relacionados com a segurança. A tecnologia de segurança pode incluir serviços subjacentes de autenticação, serviços de associação segura (distribuição de chaves, certificados, cifragens/decifragens), etc. Várias tecnologias podem fornecer estes serviços e, além disso, pode-se ainda utilizar um conjunto de interfaces de segurança genéricas, tais como GSS-API (*Generic Security Services API*), para isolar as implementações dos serviços de segurança do conhecimento detalhado dos mecanismos subjacentes.

3.4.4- Proteção Básica e comunicações

A proteção básica diz respeito ao fornecimento de meios para proteger componentes da aplicação uns dos outros, bem como proteger os componentes que suportam os serviços de segurança no ambiente operacional. Normalmente, essa proteção faz uso de serviços de *hardware* e de sistema operacional.

Se no ambiente operacional, que inclui o sistema operacional e o suporte para comunicação remota, existem serviços seguros de transporte de mensagens, a manipulação da criptografia (a partir da tecnologia de segurança) não é necessária no nível do ORB.

Em geral, deve-se estabelecer limites de proteção em torno de grupos de componentes que pertencem a um domínio de proteção. Componentes pertencentes a um mesmo domínio de proteção (mesmo sistema operacional e mesmo *hardware*) confiam uns nos outros, e as interações entre eles não necessitam ser mediadas por serviços de segurança, considerando que as interações devem estar sujeitas a controles (ver Figura 3.5a). Pode-se, ainda, fornecer meios de estabelecer relações de confiança entre componentes de domínios diferentes, permitindo que estes interajam de forma controlada entre os limites de proteção, mediante serviços de segurança (ver Figura 3.5b).

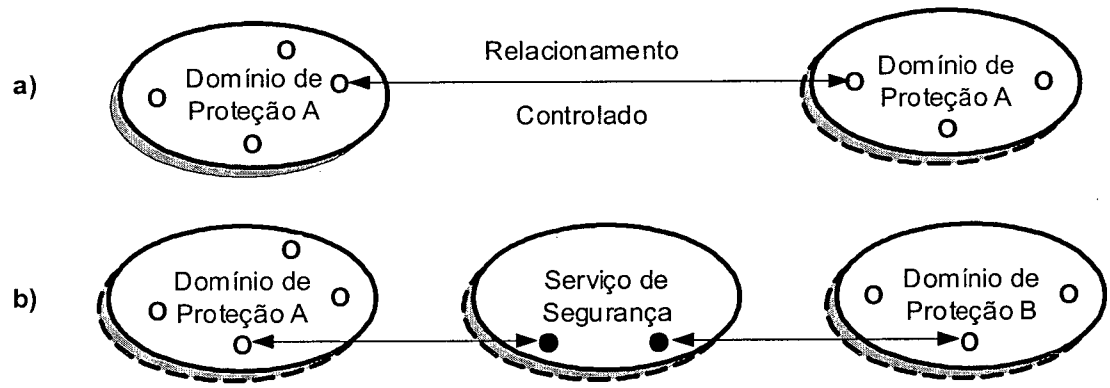


Figura 3.5 – Relações de confiança entre componentes de domínios

3.5- Autenticação de Principais

Uma entidade ativa deve primeiro estabelecer seus direitos para então tentar acessar objetos através de um ORB seguro. Um **principal** é um usuário humano ou uma entidade do sistema que é registrado *no*, e autenticado *pelo*, sistema. Este pode ser autenticado de inúmeras maneiras. A informação de autenticação mais comum para usuários humanos é uma senha, e para entidades do sistema é uma chave (*long term key*).

Se um principal, ainda não autenticado, deseja usar serviços que requerem a apresentação de privilégios autenticados, a autenticação, como apresentada na Figura 3.6, se faz necessária.

Primeiramente, o usuário fornece a sua identidade e os dados de autenticação (por exemplo, uma senha) ao ***User Sponsor***, que chama o objeto de serviço

PrincipalAuthenticator, para validar o usuário como um principal e criar o objeto ***Credentials*** que contém a identidade e os privilégios autenticados.

O procedimento de autenticação de um usuário é ativado antes que qualquer aplicação cliente seja carregada em nome do usuário. Isto permite estabelecer **credenciais** de forma transparente para as aplicações que não têm conhecimento sobre a segurança envolvida.

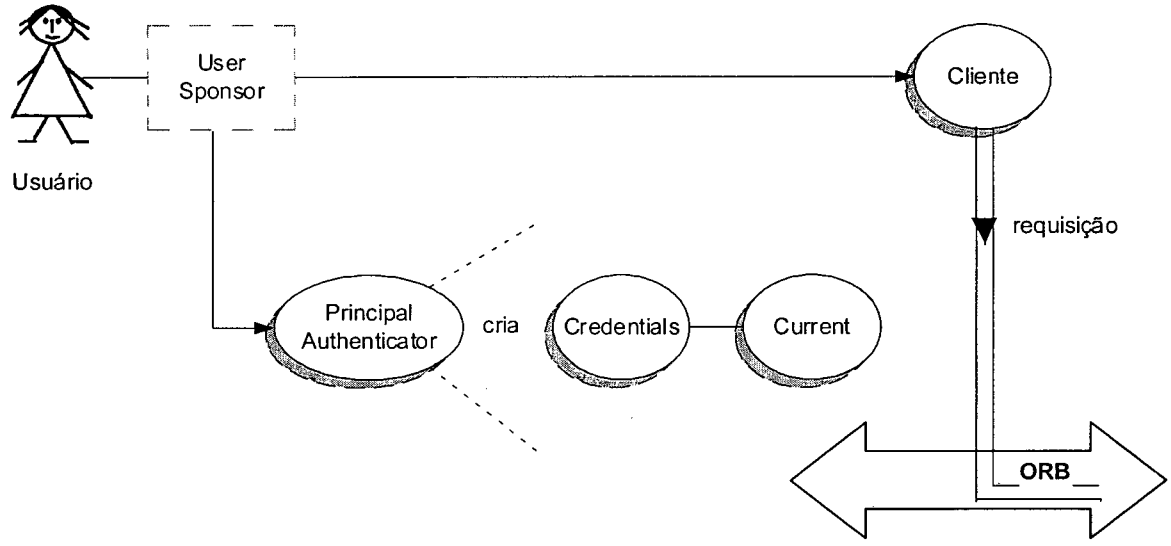


Figura 3.6 – Modelo para Autenticação

O ***User Sponsor*** representa o ponto de entrada para o usuário em um sistema seguro. É o código que chama as interfaces de segurança CORBA para a autenticação do usuário. Não precisa ser um objeto, e não tem nenhuma interface definida. É descrito para que o processo de aquisição de credenciais possa ser melhor compreendido.

O objeto ***Credentials*** armazena os atributos de segurança de um principal. Após o processo de autenticação, tem-se associado a um principal **atributos de identidade** e **atributos de privilégio**, que são mantidos em um sistema CORBA seguro em uma credencial, como mostrado na Figura 3.7. Um atributo de privilégio não precisa ser único para cada principal, podendo ser compartilhado por muitos usuários e outros principais. Esse atributo qualifica um principal. Exemplos de privilégios incluem a identidade de acesso do principal (conhecido como *AcessId*), *groups*, *roles*, *clearances* e *capabilities*.

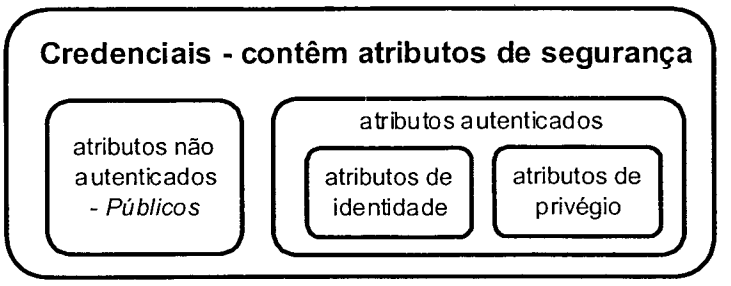


Figura 3.7 – Credenciais

O objeto ***Current*** da Figura 3.6 representa o contexto de execução atual que mantém as informações do estado de segurança dos objetos cliente e destino.

Os objetos do serviço de segurança CORBA, envolvidos no processo de autenticação (Figura 3.6), estão descritos, a seguir.

3.5.1- Objeto *PrincipalAuthenticator*

A autenticação de principais pode requerer mais de um passo, por exemplo, quando o método de autenticação *challenge/response* ou outro, com múltiplos passos, é utilizado. Por isto há as operações **authenticate** e **continue_authentication** no objeto ***PrincipalAuthenticator***. Estas operações são invocadas pelo ***User Sponsor***.

O objeto ***PrincipalAutenticator*** se encontra no pacote *SecurityLevel2* na especificação do CORBAsec.

3.5.2- Objeto Credentials

O objeto ***Credentials*** representa a informação da credencial particular de um principal. Informações, como privilégios do principal e atributos de identidade, estão incluídas neste objeto.

A *SecurityLevel2::Credentials* é a interface do programador da aplicação usada para obter os atributos de segurança pertencentes à própria aplicação e de qualquer cliente que faça invocações.

As credenciais podem ser:

- **Own_credentials**: representam as credenciais da aplicação criadas por um processo de autenticação;
- **Received_credentials**: representam o estabelecimento de um contexto de segurança entre o cliente e o objeto alvo. O objeto destino pode obter o objeto

ReceivedCredentials para identificar o pedido do principal, e obter os privilégios especiais que este pode ter.

Um objeto ***ReceivedCredentials*** representa uma informação do principal remoto para uma associação segura, por isto inclui as mesmas informações do objeto ***Credentials*** do tipo *own*, como atributos de privilégio e identidade do principal. O objeto ***ReceivedCredentials*** pode também ser usado nas invocações com delegação de privilégios.

3.5.3- Objeto *Current*

Todos os objetos de serviço de segurança são acessados através de um objeto corrente de segurança chamado ***Current*** [7]. Este objeto representa o estado associado com o fluxo de execução da invocação (*thread specific)* e armazena informações sobre o contexto corrente de segurança relativo ao processo (*capsule specific*).

Em um ambiente seguro, a interface ***SecurityLevel1::Current*** que é derivada de ***CORBA::Current*** e a interface ***SecurityLevel2::Current***, que é derivada de ***SecurityLevel1::Current***, permitem o acesso às informações de segurança associadas com o contexto de execução corrente e às credenciais associadas ao ambiente de execução. O objeto ***Current*** também pode armazenar uma credencial padrão que será utilizada apenas para fins de inicialização ou para ser utilizada por principais que não foram autenticados, mas irão utilizar serviços públicos. As operações sobre o objeto ***Current*** fornecem acesso às informações sobre os dois tipos de credenciais apresentados anteriormente.

Os serviços de segurança usam o objeto ***Current*** para obter os atributos de privilégios de um principal, obter informações relativas a uma associação segura e obter as referências dos objetos relativos ao ambiente de execução, como, o ***RequiredRights***, e o ***AccessDecision.*** Além disso, o objeto destino também pode usar o objeto ***Current*** para fornecer decisões de acesso adicionais dependentes do contexto.

3.6- Domínios e Políticas de Segurança

Um **domínio** é um conjunto de objetos que compartilham características comuns ou que respeitam um conjunto comum de regras. **Políticas de segurança** são as regras e critérios que restringem as atividades dos objetos que formam um domínio de segurança. As políticas de segurança são administradas a nível de domínios. Logo, tem-se que um

domínio também inclui um objeto, chamado *domain manager*, que gerencia todos os objetos de política do domínio em questão (ver Figura 3.8).

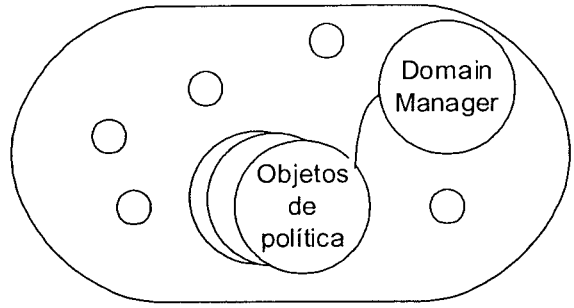


Figura 3.8 – Objetos do Domínio

O padrão CORBA de segurança define vários tipos de domínios [36]:

- **Domínios de política de segurança** (*security policy domain*). São domínios onde os objetos são governados pela mesma política de segurança. Existem vários tipos de domínios de políticas de segurança, incluindo domínios de política de controle de acesso e domínios de política de auditoria. Domínios de políticas de segurança auxiliam na resolução de problemas de gerenciamento de sistemas de objetos distribuídos;
- **Domínios de ambiente de segurança** (*security environment domains*). Compostos por objetos que possuem os requisitos da política de segurança, garantidos de forma local em seu ambiente de execução, e não por sistemas de objetos. Por exemplo, a criptografia pode não ser necessária quando as mensagens são transferidas entre objetos da mesma máquina. Dois tipos de domínio de ambiente são considerados na especificação CORBAsec: domínio de proteção de mensagem, onde a integridade e a confidencialidade estão disponíveis por alguns meios específicos, e o domínio de identidade, onde os objetos podem compartilhar a mesma identidade;
- **Domínios de tecnologia de segurança** (*security technology domains*). São domínios que usam os mesmos mecanismos de segurança para aplicar políticas. O objetivo desse domínio é identificar quais objetos usam os mesmos serviços de segurança subjacentes, tornando possível definir e manter esses serviços subjacentes e administrar as entidades da forma requerida pela tecnologia de segurança.

3.7 – Estabelecimento do Contexto de Segurança

Antes que um cliente possa usar uma referência de objeto para invocar uma operação do servidor com segurança, uma associação segura precisa ser estabelecida, associando o cliente ao objeto destino, através da referência particular do objeto. Esta associação segura é estabelecida durante o *binding* entre objetos cliente e servidor.

A especificação [36] determina que o serviço ORB, designado **interceptador de chamada segura** (*Secure Invocation Interceptor*), é responsável pelo estabelecimento do contexto de segurança requerido para proteger mensagens entre cliente e servidor, definindo o que é chamada uma associação segura. Para aplicações que não são cientes da segurança envolvida, o estabelecimento desta associação segura ocorre de forma completamente transparente.

Uma **associação segura** é estabelecida entre cliente e servidor quando :

- É estabelecida a confiança nas identidades, que envolve o servidor, autenticando os atributos de segurança do cliente e/ou o cliente, autenticando o nome de segurança do servidor;
- As credenciais do cliente (incluindo seus atributos de segurança) estão disponíveis ao servidor;
- É estabelecido o contexto de segurança que será usado para proteção das requisições de métodos e suas respostas.

A especificação define que o interceptador de chamada segura atua em **dois momentos distintos**: no instante *Bind Time* ou em tempo de *binding*, quando a associação segura é estabelecida, e no instante *Message Protection*, ou instante da proteção da mensagem.

A Figura 3.9 apresenta os serviços de segurança CORBA que participam de uma invocação segura. O interceptador de chamada segura interage com o objeto ***Vault*** que cria o objeto ***SecurityContext*** na interceptação de mais baixo nível, a nível de mensagem, na concretização da associação segura.

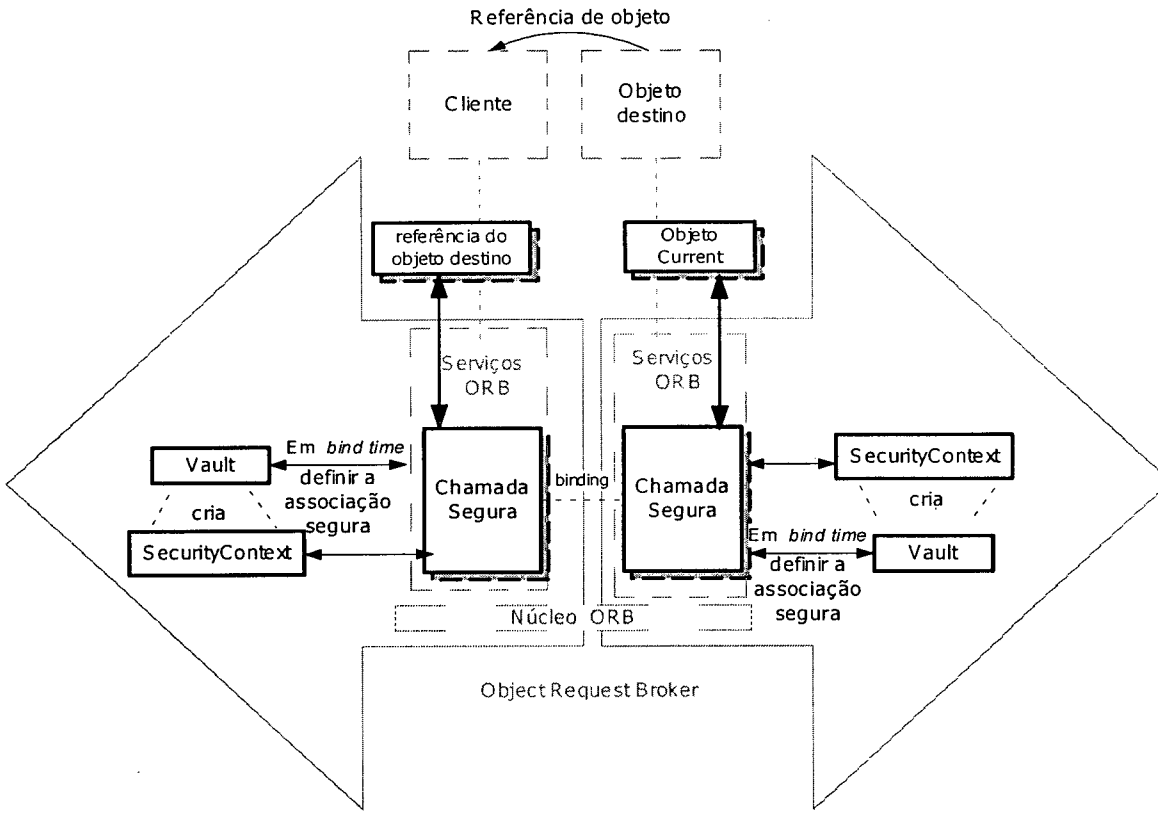


Figura 3.9 – Estabelecimento da associação segura

O objeto ***Vault*** faz parte de um módulo substituível definido no CORBAsec, (***SecurityReplaceable***) e é responsável pela negociação da associação segura. A implementação deste objeto está diretamente relacionada à tecnologia de segurança subjacente, já que estabelecer o contexto de segurança é função deste objeto.

Todo o contexto de segurança (do cliente e do servidor), que será utilizado durante a associação segura, encontra-se nos objetos ***ClientSecurityContext*** e ***ServerSecurityContext***.

3.7.1- *Bind Time* - no lado do Cliente

As funções em *bind time* do interceptor de chamada segura do cliente são usadas para:

- Encontrar quais configurações serão aplicadas para o estabelecimento da associação segura;
- Determinar algumas informações de segurança importantes para a criação do contexto de segurança, tais como, os serviços de segurança subjacentes

que serão usados, o modo de delegação usado e a credencial do cliente a ser usada;

- Estabelecer uma associação segura entre cliente e servidor. Isto é feito na primeira invocação do objeto, mas pode ser repetido quando mudanças no contexto de segurança ocorrerem;
- Tornar a informação que representa o principal (cliente) disponível ao objeto servidor.

As políticas da segurança relevantes a este interceptor são as políticas de invocação segura que constituem na configuração da associação segura. A seguir, tem-se uma descrição dos objetos que representam as políticas necessárias para a invocação.

- ***QOPPolicy***: este objeto é o portador da política de qualidade de proteção que deve ser usada para fazer uma invocação sobre uma referência do objeto.
- ***InvocationCredentialPolicy***: este objeto é portador da política que contém as credenciais que poderão ser usadas na invocação.
- ***SecurityMechanismPolicy***: este objeto é portador da política que contém os mecanismos que podem ser usados na invocação.
- ***EstablishTrustPolicy***: este objeto é portador da política que contém a orientação para o estabelecimento da confiança do cliente pelo servidor e do servidor pelo cliente.
- ***DelegationDirectivePolicy***: este objeto é portador da política que informa se a delegação pode ser usada durante a invocação.

O interceptador de chamada segura deve verificar se já existe um objeto contexto de segurança apropriado para uma associação cliente/servidor. Se um contexto apropriado já existe, ele é usado (ver seção 3.8 – Proteção da mensagem). Se não existe, o interceptador estabelece a associação segura entre cliente e servidor, criando o objeto ***ClientSecurityContext***.

3.7.2- *Bind Time* - no lado do Servidor

As funções em *bind time* do interceptor de chamada segura do servidor são usadas para:

- encontrar as políticas de invocação do servidor, registradas em sua referência de objeto;

- responder às mensagens do interceptador de chamada segura do cliente, para estabelecer a associação segura e criar o objeto ***ServerSecurityContext***.

Há uma informação de estado associada com o *binding* no cliente e no servidor. Algumas dessas informações são acessíveis através de operações no objeto ***Current*** (ver Figura 3.9).

3.8- Proteção da Mensagem

O interceptador de chamada segura é usado após o *bind time* para proteger as mensagens fornecendo integridade e/ou confidencialidade aos pedidos e respostas, de acordo com os requisitos de qualidade de proteção especificados para a associação segura no objeto ***SecurityContext*** ativo.

Uma associação segura pode, em alguns ambientes, fornecer proteção através de mecanismos inerentes ao ambiente, como por exemplo, empregando uma camada de proteção abaixo da camada de mensagem do ORB. O ORB não necessita garantir sua própria proteção de mensagem quando este opera sobre uma camada de transporte segura (p.ex., o SSL).

Como o CORBAsec tem que atender a uma variedade de propósitos e níveis de proteção, as interfaces apresentadas na especificação permitem a escolha de algoritmos criptográficos para fornecer proteção. A qualidade da proteção depende da qualidade dos mecanismos subjacentes.

3.9- Autorização e Controle de Acesso

Depois de serem autenticados, precisa-se ter a garantia de que os clientes irão apenas executar operações a eles autorizadas. Para isto, deve-se aplicar o processo de autorização. O modelo CORBA de segurança pode construir políticas de autorização usando mecanismos de listas de controle de acesso (**ACLs** – *Access Control Lists*), listas de *capability*[8] e controle de acesso baseado em *roles*[9] (**RBAC** – *Role Based Access Control*). A seguir será descrito o modelo global usado para todos os tipos de controle de acesso (ver Figura 3.10).

[8] É um atributo de privilégio que pode ser traduzido como competência.
[9] É um atributo de privilégio que representa a posição ou função de um usuário no momento de sua autenticação.

Observa-se na Figura 3.10 que dois tipos de políticas de controle de acesso podem ser aplicadas: uma política de acesso que pode ser aplicada pela própria aplicação cliente (*application object access policy*), sendo que para isto a aplicação tem que estar ciente da segurança, e uma política de acesso que é aplicada pelo próprio ORB e pelos serviços de segurança que o ORB usa para as aplicações cientes ou não da segurança (*object invocation access policy*).

O modelo de autorização especificado no CORBAsec está baseado no uso de **funções de decisão de acesso** (objetos ***AccessDecision***), que determinam se uma operação pode ser executada, conforme apresentado na Figura 3.11.

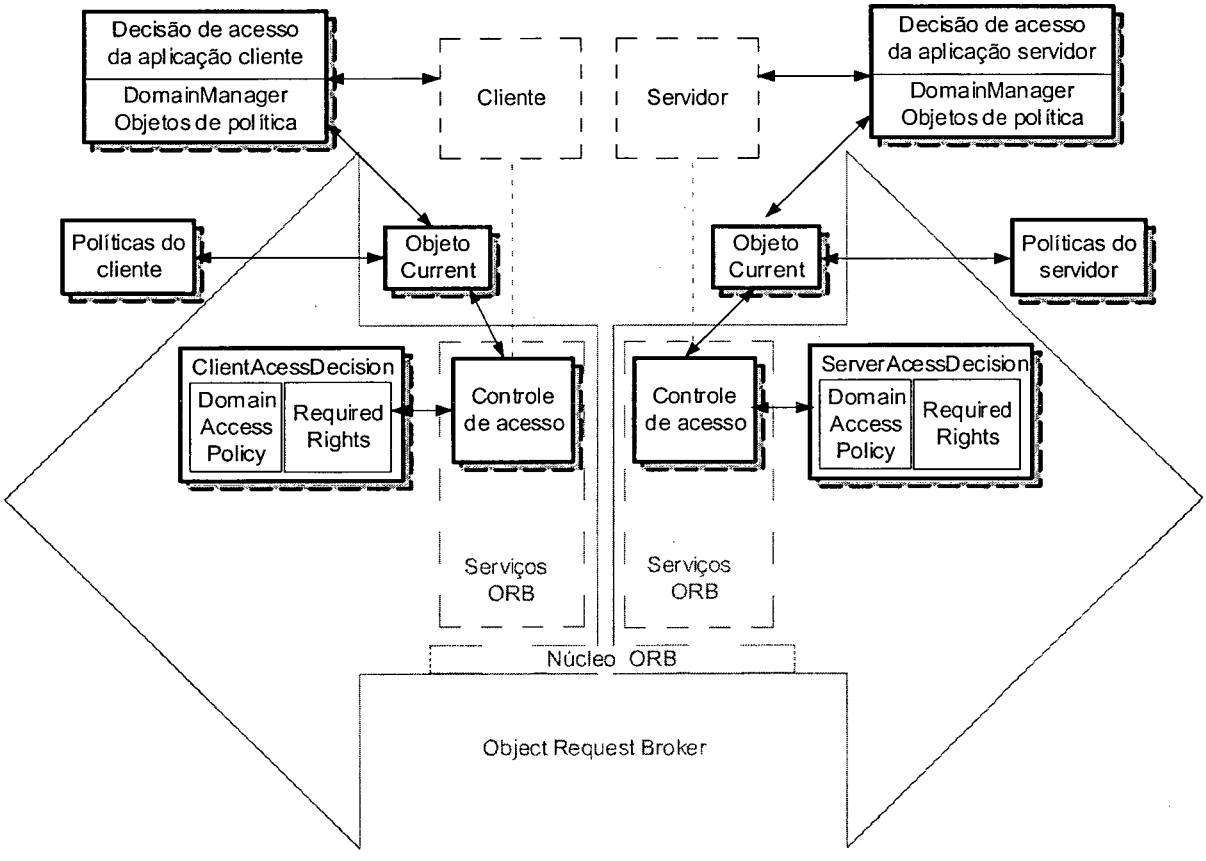


Figura 3.10 – Modelo de Controle de Acesso do CORBAsec

As políticas de acesso específicas da aplicação (*application object access policy*) precisam definir suas próprias funções de decisão de acesso para garantir o cumprimento das suas próprias regras de controle de acesso. A especificação não define interfaces administrativas próprias para esses tipos de políticas, visto que aplicações diferentes

possuem necessidades diferentes. Os objetos cliente e destino podem chamar um objeto ***DomainManager*** para aplicar políticas de acesso local.

Nas políticas de acesso aplicadas na invocação dos objetos (*object invocation access policy*), a nível de *middleware*, o interceptador de controle de acesso chama o objeto ***AccessDecision*** que decide, de acordo com os objetos ***RequiredRights*** e ***DomainAccessPolicy***[10], se a invocação deve ou não ser permitida.

A especificação CORBAsec define que o interceptador controle de acesso atua em dois momentos distintos: no instante *bind time*, ou, em tempo de *binding*, quando este interceptador obtém as políticas que serão usadas na invocação, e no instante *access decision time*, ou no instante da decisão de acesso, quando o objeto ***AccessDecision*** decide se o pedido pode ser atendido ou não.

Após o *bind time*, o objeto ***Current*** contém referências de importantes objetos usados para garantir a política de autorização, como as referências dos objetos ***RequiredRights*** e ***AccessDecision***.

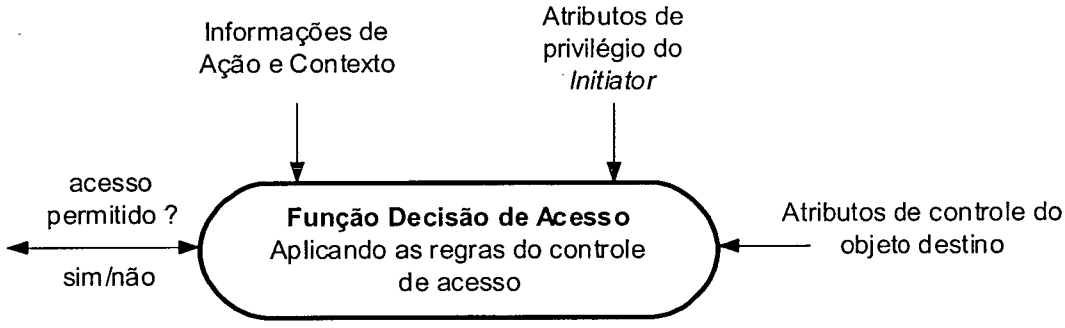


Figura 3.11 – Modelo de Autorização

As funções de decisão de acesso no lado do cliente definem as condições que permitem ao cliente invocar uma operação sobre o objeto destino. Funções de decisão de acesso no lado do servidor definem as condições que permitem ao objeto destino aceitar a invocação. Uma ou ambas as funções de decisão podem existir.

A política de acesso é construída dentro dessas funções de decisão de acesso, que retornam sim ou não, quando solicitadas para checar se o acesso é permitido.

[10] Para relembrar, o objeto ***DomainAccessPolicy*** representa a interface de gerenciamento de políticas de autorização discricionárias e concede a um conjunto de principais direitos específicos para executar operações sobre todos os objetos do domínio, e o objeto ***RequiredRights*** armazena as informações sobre os direitos necessários para executar cada método de cada interface do sistema.

A função decisão de acesso usada na invocação do objeto baseia-se:

- nos atributos de privilégio do principal/cliente;
- em qualquer controle sobre estes atributos, por exemplo o tempo em que eles são válidos;
- na operação a ser usada;
- nos atributos de controle do objeto destino.

Um principal pode ter uma variedade de atributos de privilégio usados para controle de acesso, tais como: identidade de acesso do principal; *roles*, que geralmente estão relacionados às funções desempenhadas por um usuário; grupos, que normalmente refletem uma estrutura organizacional; habilitação de segurança (*security clearance*); *capabilities;* e outros privilégios que uma empresa define como úteis para controlar o acesso.

Os atributos de controle, associados ao objeto destino, podem ser: listas de controle de acesso que identificam os usuários permitidos pelo nome ou por outros atributos de privilégio; e informações usadas em esquemas baseados em rótulos(*label-based schemes)*, como a classificação de um objeto que identifica a habilitação (*clearance*) de segurança dos principais permitidos para executar operações.

Um sistema de objetos pode ter uma grande quantidade de objetos, cada qual com suas várias operações. Portanto, não é prático e eficiente associar atributos de controle com cada operação em cada objeto. Isso acarretaria uma alta sobrecarga para a administração do sistema e dificuldade para armazenar as informações. Consequentemente, espera-se que os atributos de controle sejam compartilhados por categorias de objetos, particularmente por objetos do mesmo tipo no mesmo domínio de política de segurança.

Atributos de controle podem ser associados a um conjunto de operações de um objeto e não somente a uma operação individual. Consequentemente, um usuário com privilégios especificados pode ter direitos (*rights*) para invocar um conjunto específico de operações.

As políticas de autorização, usando atributos de controle, podem: agrupar sujeitos usando privilégios; objetos, usando domínios; e operações, usando direitos.

A especificação CORBA de segurança provê suporte direto a políticas de autorização discricionárias através do objeto ***DomainAccessPolicy***, usa o modelo de

controle de acesso com ACLs para armazenar os direitos de acesso, e utiliza uma interface de gerência associada a essas políticas (***SecurityAdmin***).

3.10- Auditoria

Serviços de auditoria permitem ao sistema gerenciar o monitoramento de eventos do ORB, como por exemplo, tentativas de violação de segurança através de registros dos detalhes de eventos relevantes à segurança do sistema. Um grande número de eventos pode ser registrados. Entretanto, políticas de auditoria são usadas para restringir quais os tipos de eventos devem ser examinados e em quais circunstâncias.

De acordo com a Figura 3.12, que apresenta o modelo do serviço de auditoria do modelo CORBAsec, têm-se duas categorias de políticas de auditoria:

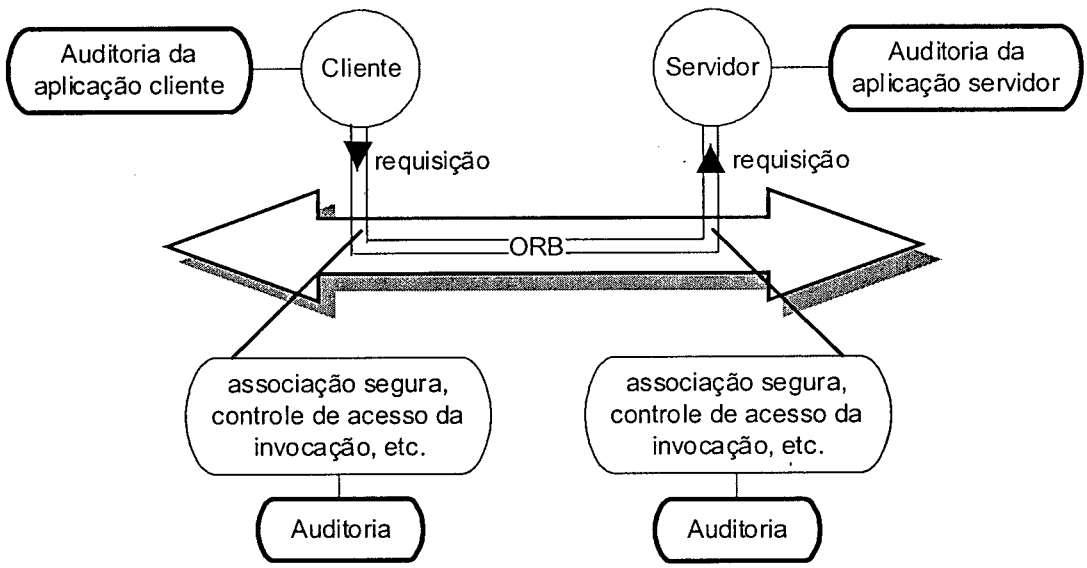


Figura 3.12 – Modelo do serviço de auditoria

- As **políticas de auditoria do sistema** que controlam quais eventos são registrados como resultado das atividades relevantes do sistema, como autenticação, mudanças de privilégio, sucesso ou falha de invocações de objetos e administração de políticas de segurança. Estas políticas são aplicadas automaticamente pelo ORB para todas as aplicações (cientes ou não da segurança).
- As **políticas de auditoria da aplicação** que controla quais eventos são examinados pelas aplicações cientes da segurança.

Os eventos podem ser registrados sobre rastros de auditoria (*audit trails*) para depois serem analisados ou surgem alarmes que podem ser enviados aos administradores. A especificação CORBAsec apresenta como os registros de auditoria são gerados e escritos em canais de auditoria, mas não especifica como esses registros são filtrados depois, como os canais e os rastros da auditoria são mantidos seguros e como os registros podem ser coletados e analisados.

3.11- Não-Repudiação

A funcionalidade de segurança opcional da não-repudiação garante que um objeto que realizou uma operação no sistema, como por exemplo, a emissão de uma mensagem, não possa negar futuramente que executou essa operação. Esses controles de não-repudiação não são feitos automaticamente a cada invocação de método, mas é a própria aplicação (ciente da segurança) que deve efetuá-los chamando o objeto ***NRCredentials*** para gerar e verificar a evidência. A noção de **evidência** é definida no CORBA 2.0 como a prova de que uma ação foi realizada.

Os dois tipos de provas que são abordados no CORBA 2.0 são a prova de criação de mensagem (*proof of the origin*) e a prova de recebimento de mensagens (*proof of receipt*). O *framework* de não-repudiação está baseado no modelo ISO de não-repudiação

3.12- Gerência dos domínios e das Políticas de Segurança

As duas principais tarefas do administrador do sistema são as gerências dos domínios de segurança e das políticas de segurança. A gerência dos domínios tem por objetivo definir e criar os domínios, definir servidores de domínios, quando necessários, e gerenciar os membros dos domínios.

A gerência das políticas de segurança visa especificar as políticas, definir opções da política (seguindo os mecanismos de segurança subjacentes), especificar quais políticas se aplicam a quais domínios, dentre outras tarefas. A especificação CORBAsec inclui um *framework* para a administração das políticas de segurança e detalha como administrar tipos de políticas particulares. Por exemplo, este *framework* inclui operações para especificar qualidade de proteção padrão para as mensagens do domínio e eventos para auditoria.

A especificação atual do CORBASec [36] não determina os procedimentos padrões para gerenciar os membros dos domínios de políticas de segurança. Entretanto, existe um trabalho da OMG chamado RFP/orbos-9812-24 (*Security Domain Membership Management Service*), que tem como proposta padronizar interfaces de serviços comuns para gerenciar domínios de políticas de segurança e os membros dos domínios de políticas de segurança.

3.13- Considerações gerais sobre a segurança CORBA

O modelo de segurança do padrão CORBA pode ser visto sob cinco diferentes pontos de vista:

- Da **administração do empreendimento**, responsável pelos recursos empresariais, incluindo os sistemas de tecnologia da informação, que visam prevenir as ameaças aos recursos, a um custo disponível ao administrador do empreendimento. Por conhecer a estrutura organizacional, o administrador deve então identificar os diferentes domínios e definir onde as diferentes políticas de segurança devem ser aplicadas.
- Do **usuário final,** pois o usuário humano é um indivíduo que normalmente precisa ser autenticado. Um usuário é modelado no sistema como um principal que pode ter atributos de privilégios, como *roles* ou grupos, válidos na sua organização.
- Do **desenvolvedor da aplicação**, responsável pelos objetos das aplicações, cuja principal preocupação é executar as funções empresariais. Muitos desenvolvedores de aplicações podem não estar cientes da segurança do sistema, embora estejam sendo protegidos por este. Entretanto, alguns desenvolvedores, necessitam garantir aos objetos da aplicação alguma segurança extra, por exemplo, possibilitando a esta definir a qualidade da proteção para invocação de objetos e auditoria executada pela própria aplicação.
- Da **administração**, pois os administradores entendem mais do sistema que os usuários, considerando que o administrador conhece como estes estão agrupados em domínios de políticas e quais privilégios devem ser atribuídos a

eles no processo de autenticação. Os administradores devem ser responsáveis pela criação e manutenção dos domínios, especificando quem deve ser membro deste domínio, e pela administração das políticas de segurança.

- Da **implementação** de sistema de objetos, pois os desenvolvedores de sistemas seguros devem reunir em um ORB outros serviços de objetos e/ou facilidades comuns e serviços de segurança, requeridos para fornecer as características desejáveis de segurança. O implementador do ORB em um sistema de objetos seguros pode usar serviços de segurança ORB durante a invocação de objetos e limites de proteção para prevenir interferências entre objetos.

A especificação do serviço de segurança CORBA define um modelo para **segurança da interoperabilidade** entre ORBs, conforme as especificações do CORBA 2.0 que enfatizam a necessidade de uma mesma tecnologia de segurança nos serviços subjacentes. Na arquitetura de interoperabilidade do CORBA 2.0, para fornecer uma forma flexível de interoperabilidade segura, um novo protocolo foi inserido abaixo do protocolo GIOP (ver Figura 3.13). O protocolo, que garante a segurança da transmissão das mensagens GIOP e que suporta o objeto de serviço ***SecurityContext*** do modelo CORBAsec para aplicar esta proteção de mensagens, é o SECIOP (*Secure Inter-ORB Protocol*).

A especificação de interoperabilidade CORBA suporta ainda o protocolo DCE-CIOP (*Distributed Computing Environment – Common Inter-ORB Protocol*) que se baseia nos serviços de segurança DCE e no RPC autenticado DCE.

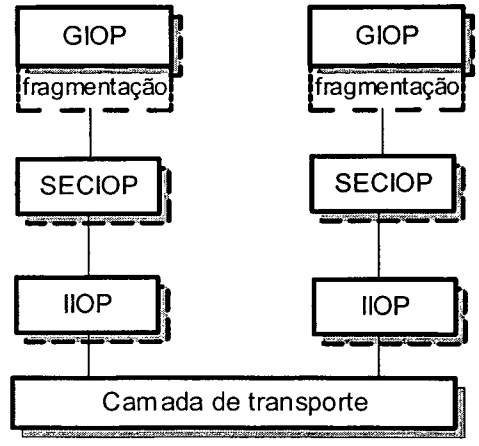


Figura 3.13- Protocolo SECIOP

Para definir o nível de interoperabilidade, as tecnologias de segurança podem assumir três níveis CSI (*Common Secure Interoperability*), de acordo com o tipo de política de acesso no qual este está baseado e as facilidades de delegação suportadas. As tecnologias, que podem ser utilizadas para fornecer os serviços subjacentes no modelo do CORBASec, estão descritas a seguir:

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

3.14- Levantamento Bibliográfico de Experiências de ORBs seguros

Uma experiência acadêmica, desenvolvida pela Universidade de Illinois (EUA) de uma implementação em Java do modelo de segurança CORBA, pode ser encontrada em *Nephilim: Java Implementation of CORBA Security Services* [28]. Este trabalho fornece

um pacote chamado JGSS[11] para suportar autenticação baseado no SESAME e protocolos de delegação. Este pacote fornece **CSI-ECMA** (baseado no SESAME e no mecanismo ECMA GSS-API), suportando o CSI nível 2, que é usado como tecnologia de segurança. Usando interfaces de interceptdores padrão, esta experiência visa implementar controle de acesso a nível de ORB e objetos de delegação e auditoria segura conforme especificado no serviço de segurança CORBA, necessários para garantir o nível 2 de funcionalidade de segurança. Outro projeto da Universidade de Illinois, chamado *Malakin*, procura soluções para o problema de integridade e interoperabilidade em ambientes de objetos distribuídos.

O projeto *JacORBoverSSL*[12], desenvolvido por Andre Benvenuti, é uma implementação em Java que visa integrar um protocolo SSL ao JacORB. Como esta implementação do IIOP sobre o SSL foi realizada no núcleo do ORB, a tecnologia de segurança introduzida não está de acordo com os níveis do modelo de segurança CORBA.

Além das experiências acadêmicas citadas acima, existem alguns produtos comerciais que fornecem ORBs seguros. Podem ser destacados: *Jbroker* [31], *LiveContent BROKER* [40], *ORBAsec SL2* [1], *VoyagerSecurity* [32], *OrbixSecurity* [18] e *SecureBroker* [41]. Uma breve análise desses ORBs é apresentada, a seguir.

O ORB seguro ***LiveContent BROKER*** fornece autenticação e garante associações seguras, usando como mecanismo de segurança subjacente o SESAME ou o Kerberos. As leituras realizadas nas documentações do ***LiveContent BROKER*** não permitiram verificar se este segue o modelo CORBAsec.

Analisando a documentação do ***OrbixSecurity*** [18], verifica-se que a implementação deste ORB seguro atende ao nível 2 de funcionalidade de segurança do CORBAsec. Três mecanismos de autenticação de principais são suportados: *user ID* e *password login*; *login* usando *Security Dynamic' SecurID*; e esquema de autenticação baseado em chave pública. Este produto fornece um controle de acesso distribuído, incluindo um controle a nível de aplicação com o auxílio de Certificados de Atributos de Privilégios (PACs) usando o padrão X500 e de certificados SSL (para os atributos de identidade). A segurança da comunicação (integridade e confidencialidade) é garantida pelo protocolo SSL, usado como mecanismo de segurança subjacente. Auditorias do processo de autenticação, estabelecimento da comunicação e invocação de objetos podem

[11]Pacote Java que implementa uma interface de segurança genérica (GSS-API) especificada na RFC-1508.

ser realizadas por este ORB seguro. O *OrbixSecurity* implementa as interfaces de administração de segurança conforme requeridas pelo nível 2 do CORBAsec. Constatou-se que o *OrbixSecurity* segue a maioria das interfaces definidas no CORBAsec, apesar de usar algumas funcionalidades que ainda não foram adicionadas na especificação, mas que possivelmente serão (p.ex., o uso de PAC, gerência de domínios e algumas modificações no processo de auditoria). Além disso, o OrbixSecurity não implementa interceptadores padrão. Os mecanismos de desvio para controle de acesso são implementados através de filtros (*filters*), presentes no *Orbix*. Outro tipo de interceptador presente no *OrbixSecurity* é o *transformador* (*transformer*) que diferentemente do filtro, pode operar sobre os pedidos. Neste produto, *transformadores* são usados na interceptação de baixo nível.

O **Jbroker** da ObjectEra Inc fornece o suporte para o uso do IIOP sobre SSL como definido na especificação *CORBAsec*. Entretanto, os processos de autenticação e controle de acesso não seguem os modelos definidos na especificação do serviço de segurança CORBA.

O ORB seguro da Object Space (**VoyagerSecurity**) fornece um *framework* de segurança que garante a integridade e confidencialidade dos dados com o uso do protocolo SSL. A integração desta tecnologia, bem como os serviços de autenticação e controle de acesso, não estão de acordo com os modelos e interfaces definidos no *CORBASec*.

De acordo com [1] e [41], o **ORBAsec SL2** fornecido pela Adiron e o **SecureBroker** da Promia Incorporated cumprem perfeitamente o modelo do CORBASec. Ambos usam o Kerberos e o SSL como tecnologias de segurança. Autenticação e associações seguras são fornecidas por esta implementação. O ORBAsec SL2 não atende totalmente o nível 2 de funcionalidade de segurança CORBA, mas segue totalmente a especificação CORBAsec. Os objetos ***RequiredRights***, ***DomainAccessPolicy***, ***AccessDecisiom*** e ***AuditingDecision*** não são usados nesta implementação.

O SecureBroker ainda está em fase de desenvolvimento, somente a integração com o SSL está disponível (SecureBrokerSSL). O *SecureBroker CORBA Security 2.0 Java version* empregará o Kerberos/SESAME, que utilizará o JacORB (ORB desenvolvido na Universidade de Berlim)[6] e fornecerá comunicação criptografada, controle de acesso e auditoria.

---

[12] Em breve, esta implementação será incorporada e distribuída no JacORB v1.0

Para uma melhor ilustrar as características dos produtos comerciais descritos acima, elaborou-se o quadro comparativo apresentado na Tabela 3.1.

3.15- Conclusões do Capítulo

As boas características do CORBA, como reusabilidade, portabilidade e interoperabilidade de *softwares*, orientados a objetos, o tornam um importante e respeitado padrão para integração de aplicações distribuídas. Além disso, a plataforma CORBA está se constituindo um padrão de *facto* para programação distribuída na Internet. Entretanto, algumas aplicações de sistemas de larga escala exigem uma característica a mais: a segurança. Em resposta a esta necessidade, a OMG adicionou o serviço de segurança CORBA à especificação *CORBAservices*.

Este capítulo apresentou brevemente os conceitos e a arquitetura CORBA, visando facilitar o entendimento do serviço de segurança CORBA, objetivo principal deste capítulo.

A especificação do serviço de segurança do CORBA é a maior e mais complexa especificação produzida pela OMG e abrange uma ampla série de requisitos de segurança [16]. Algumas abstrações dessa especificação podem levar a uma interpretação errônea dos objetos de segurança CORBA, bem como dos objetos de segurança do ORB ou das funcionalidades de segurança do modelo CORBAsec. Procurou-se neste capítulo descrever o modelo estrutural e as principais funcionalidades de segurança (autenticação, esquema de autorização, estabelecimento de uma associação segura), destacando o papel de cada objeto envolvido e o processo para garantir as funcionalidades. Foram abordados ainda neste capítulo, aspectos relativos a delegação, domínios, gerência das políticas de segurança, auditoria, não-repudiação, e a interoperabilidade segura descrita no CORBA 2.0

ORBs SEGUROS	CARACTERÍSTICAS DE ACORDO COM A ESPECIFICAÇÃO CORBAsec					
	Autenticação	Associação segura	Controle de acesso	Interceptadores padrão	Nível de funcionalidade	Outros
Jbroker	Não segue a especificação	Garantida pelo SSL	Não segue a especificação	?	?	-
LiveContentBroker	Kerberos/ SESAME	?	?	?	?	-
ORBAsec SL2	Kerberos	Garantida pelo Kerberos e SSL	Não fornece	Não implementa	Nível 2 (parcialmente)	-
VoyagerSecurity	Não fornece	Não segue a especificação	Não fornece	?	Não segue a especificação	-
OrbixSecurity	Mecanismos de autenticação subjacentes	Garantida pelo SSL	A nível de aplicação e a nível de *midleware*	Não implementa	Nível 2	Auditoria a nível de *midleware*
SecureBroker (em desenvolvimento)	Kerberos/SESAME	Kerberos / SESAME e SSL	A nível de aplicação e a nível de *midleware*	Implementa	Nível 2	Auditoria a nível de *midleware*

Obs: Usou-se o símbolo ? quando as definições constantes da bibliografia consultada não possibilitaram a verificação da compatibilidade da característica com a especificação CORBAsec

Tabela 3.1 – Quadro Comparativo de ORBs seguros

Capítulo 4: Esquema de autorização discricionária baseado no CORBAsec

4.1- Introdução

O padrão CORBA de segurança é muito extenso e há pouca bibliografia que descreva seus principais aspectos de forma clara e sucinta, demonstrando seu uso em aplicações práticas de sistemas de larga escala. Atualmente, o número de empresas que desenvolve ORBs seguros ainda é pequeno. De igual forma, o CORBAsec ainda é pouco explorado pela comunidade científica brasileira. O esquema de autorização discricionária, descrito neste capítulo, visa verificar a aplicabilidade das abstrações do modelo CORBAsec, esclarecendo os aspectos relacionados à dinâmica do modelo, tendo em vista o desenvolvimento de uma aplicação real. Esse esquema faz parte de um projeto maior que visa desenvolver um esquema de autorização com políticas discricionárias e obrigatórias para redes de larga escala, envolvendo modelos e ferramentas de programação representados pelo Web, Java e CORBA: o ***JaCoWeb Security***[1].

Um refinamento das abstrações do serviço de segurança CORBA é apresentado visando descrever um esquema de autorização discricionária para sistemas abertos, totalmente baseado nas interfaces definidas no CORBAsec. Em seguida, se introduziu o protótipo implementado neste trabalho. A conformidade do protótipo com a especificação do serviço de segurança do CORBA é analisada visando verificar a aplicabilidade das abstrações do CORBAsec.

[1] A descrição deste projeto pode ser encontrada em [50], [51] ou ainda na página do projeto (http:\\www.lcmi.ufsc.br/jacoweb)

4.2- Refinamento da especificação CORBAsec

A especificação CORBAsec define que uma credencial é composta por uma lista de atributos de segurança (**SecAttribute[]**). Cada atributo é definido por um tipo de atributo de segurança (estrutura **AttributeType**), uma autoridade que valida este atributo (*defining_authority*, do tipo *byte[]*) e o valor do atributo (*value*, do tipo *byte[]*). A estrutura **AttributeType** é formada pelo objeto **ExtensibleFamily** que define a família do atributo e por um elemento inteiro que identifica o tipo do atributo (*attribute_type*). O objeto **ExtensibleFamily** é composto por um identificador da autoridade que define o atributo (*family_definer*) e pelo identificador da família (chamado *family*). A OMG reserva o valor zero para o definidor da família CORBA (ver Tabela 4.1) e define um número de constantes para o campo *attribute_type* da estrutura **AttributeType**, definido para a família CORBA (ver Tabela 4.2). A especificação CORBAsec não define nenhum padrão para os campos *defining_authority* e *value* dos atributos definidos na Tabela 4.2 [1].

family definer	*CORBA family*	Descrição
Família CORBA (0)	0	Atributos de identidade
	1	Atributos de privilégio

Tabela 4.1- Família CORBA

CORBA Family	Attribute_type	Descrição
0	1	AuditId
	2	AccountingId
	3	NonRepudiationId
1	1	Public
	2	AccessId
	3	Primary
	4	GroupId
	5	**Role**
	6	AttributeSet
	7	Clearance
	8	Capability

Tabela 4.2- Tipos de atributos de segurança da família CORBA

Para políticas de segurança que usam os privilégios atribuídos a um principal para decidir se este possui os direitos necessários para executar uma determinada operação, o CORBAsec oferece oito tipos de atributos de privilégio.

Conforme descrita na seção 2.3, a definição de política de segurança ou política de autorização está fortemente vinculada às questões: *o que* se deseja proteger, e *como* aplicar esta proteção. As políticas de segurança devem ser adotadas e projetadas para assegurar níveis de segurança apropriados às atividades que são executadas no sistema (no contexto deste trabalho, têm-se os sistemas de larga escala); e, os mecanismos de segurança devem ser empregados para concretizar essas políticas de segurança estruturando um **esquema de autorização**.

A implantação de um esquema de autorização deve, além de aplicar um mecanismo de controle de acesso, utilizar controles internos para concretizar as políticas de segurança. Os serviços de identificação e de autenticação, que fornecem a confiança necessária entre as partes envolvidas em uma associação e os controles criptográficos, que garantem a proteção das mensagens transmitidas nessa associação, são exemplos desses controles internos.

O esquema de autorização, aqui apresentado, está baseado no modelo de matriz de acesso introduzido por Lampson [19] e descrito para políticas de autorização discricionárias. Nessas políticas, define-se que os direitos de acesso a cada recurso são manipulados livremente pelo responsável do recurso (geralmente o proprietário do mesmo), segundo a sua vontade.

A Figura 4.1 ilustra a estrutura geral do esquema de autorização discricionária, apresentando os objetos definidos no modelo de segurança CORBA necessários para aplicação das funcionalidades de segurança tratadas no esquema. É preciso ressaltar que o modelo de segurança a ser discutido neste capítulo considera que a segurança é garantida de forma transparente à aplicação; sendo assim, as aplicações consideradas não são cientes da segurança (*unaware*)

Através dos objetos ***User Sponsor*** e ***PrincipalAuthenticator*** apresentados na Figura 4.1, pretende-se identificar e autenticar os usuários que desejam, através de um objeto de aplicação cliente, solicitar algum tipo de serviço a um objeto cuja referência pode ser obtida a partir do Serviço de Nomes CORBA. Como resultado desta autenticação, têm-se definidas as credenciais deste cliente. Em baixo nível, com os objetos ***Vault*** e

SecurityContext, espera-se estabelecer e usar um contexto de segurança que garanta integridade e confidencialidade da comunicação (ver Figura 4.1).

A política de autorização discricionária, baseada em listas de controle de acesso (ou ACLs), é aplicada com base nos objetos ***RequiredRights*** e ***DomainAccessPolicy*** (ver Figura 4.1). O esquema de autorização garante que somente usuários autorizados poderão solicitar serviços a uma aplicação.

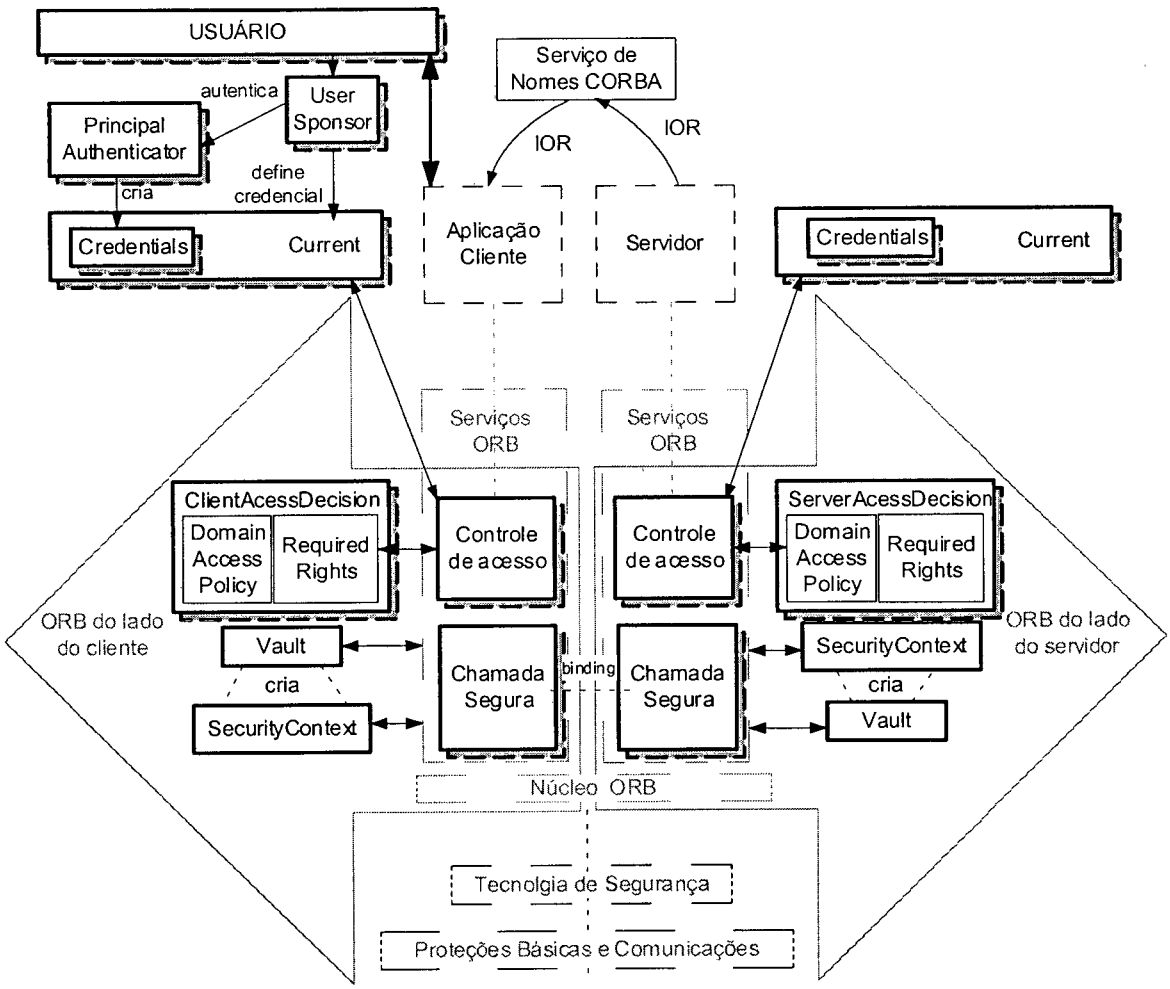


Figura 4.1- Esquema de Autorização Discricionária

Os detalhes da dinâmica do esquema da Figura 4.1 encontram-se descritos a seguir:

4.2.1- Dinâmica da Autenticação

Para proteger informações críticas, um sistema de informação precisa identificar quem está tentando acessar suas informações. Como apresentado na seção 3.5, a especificação do serviço de segurança CORBA descreve um modelo e define interfaces para que os objetos desse modelo executem a autenticação de principais. A Figura 4.2,

baseada na especificação CORBAsec, procura apresentar o serviço de identificação e autenticação de usuários proposto no projeto *JaCoWeb Security*.

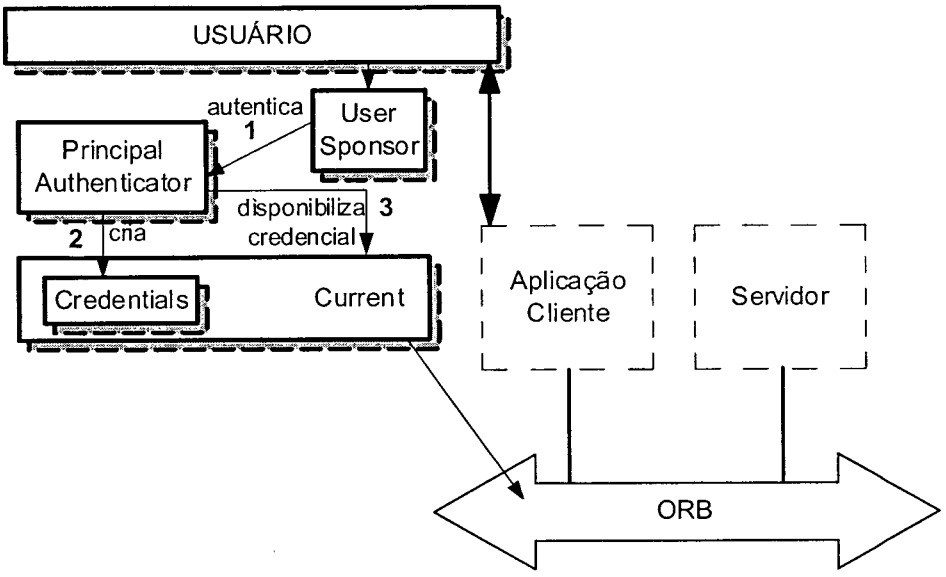


Figura 4.2- Dinâmica da Autenticação de Principais

Um principal deve estabelecer suas credenciais antes de invocar operações a um objeto seguro. Muitos clientes possuem credenciais do tipo padrão, criadas quando o usuário entra no sistema. Estas credenciais são automaticamente usadas na invocação do objeto sem que o cliente execute alguma ação específica. Mesmo se a autenticação de usuários é executa dentro do sistema de objeto, feita normalmente por um ***User Sponsor*** (p.ex., programa de *login*), a aplicação cliente pode permanecer não ciente da segurança.

No passo 1 da Figura 4.2, através do método **authenticate** da interface ***PrincipalAuthenticor***, o ***UserSponsor*** solicita que uma credencial, contendo os atributos de segurança de um principal, seja criada

Para que as credenciais sejam criadas, o ***UserSponsor***, ao invocar o ***PrincipalAuthenticator*** (pelo método **authenticate**), precisa fornecer o método de autenticação que deve ser usado, o mecanismo de segurança que criará as credenciais, a informação de identificação do principal (p.ex., nome de *login*), a informação de autenticação do principal (p.ex., uma senha) e os atributos de privilégios que o principal deseja autenticar (p.ex., ser membro de um grupo ou ter autorização para um *role* específico).

O ***PrincipalAuthenticator*** pode usar outros objetos para autenticar o usuário e estabelecer seus privilégios, por exemplo, um servidor de autenticação e/ou um servidor de

atributos de privilégio [39]. O servidor de autenticação possui o papel de verificar a identidade do principal de acordo com o dado de autenticação fornecido por este, podendo suportar: um esquema de autenticação baseado em um identificador de usuário e senha (*password*) normal; uma autenticação fornecida pelo protocolo Kerberos[2]; ou um esquema de autenticação baseado em chave pública (PKI – *Public Key Infrastruture*). O servidor de atributos de privilégio pode ser usado para gerenciar os privilégios de acessos de principais e para distribuir Certificados de Atributos de Privilégio (PAC – Privilege Attribute Certificates)[3] que possibilitam a transmissão de privilégios sob uma associação segura.

No passo 2 da Figura 4.2, depois da operação **authenticate** ou **continue_authentication** (quando múltiplos passos para a autenticação se fazem necessários) retornar o valor de autenticação "*SecAuthSucess*", que indica que a autenticação foi concluída com sucesso, o objeto ***PrincipalAuthenticator*** cria o objeto ***Credentials*** e o coloca na lista **own_credentials** do objeto ***Current*** (passo 3).

Para suportar diferentes serviços de autenticação, sem necessitar substituir o objeto ***PrincipalAuthenticator***, a criação de credenciais pode passar a ser feita pelo objeto ***Vault*** (módulo *SecurityReplaceable*) através de interfaces de segurança genéricas (GSS-APIs) que são independentes das tecnologias de segurança. A especificação do CORBAsec, já prevendo esta funcionalidade extra para o objeto ***Vault***, definiu o método **acquire_credentials** que deve ser invocado pelo ***PrincipalAuthenticator*** para que o objeto ***Vault*** assuma as suas funções de autenticação.

4.2.2- Dinâmica das Atividades Administrativas

O modelo administrativo descrito na especificação do CORBAsec sugere como administrar as políticas de segurança. As atividades administrativas descritas na especificação são as seguintes [36]:

- criar objetos em um ambiente seguro sujeito às políticas de segurança;
- encontrar os gerentes de domínios aplicados às políticas de segurança;
- encontrar as políticas pelas quais os gerentes de domínios são responsáveis;

---

[2] O Kerberos é um protocolo de autenticação baseado em chave privada, desenvolvido no MIT (*Massachusetts Institute of Technology*) e padronizado pela organização IETF (*Internet Engineering Task Force*).

[3] O uso do *Privilege Attribute Certificate* está sendo proposto pela OMG no *Request For Proposal (RFP) Common Secure Interoperability Version 2* (OMG Document orbos/99-01-10). O padrão PAC é independente do mecanismo de segurança subjacente.

- especificar quais direitos dão acesso a quais informações, para suportar políticas de autorização.

Segundo o modelo CORBAsec, quando um objeto CORBA é registrado em um serviço de nomes e se torna visível via CORBA, automaticamente, ele também torna-se membro de um ou mais domínios de políticas de segurança. Entretanto, as especificações atuais do CORBAsec não indicam os procedimentos para gerenciar membros dos domínios e não definem procedimentos para incluir ou remover políticas [36][38].

Não faz parte do esquema de autorização introduzido neste capítulo o tratamento das atividades administrativas descritas acima. Entretanto, o gerenciamento da política de autorização, bem como o serviço de política que tem como objetivo possibilitar o gerenciamento centralizado de objetos de política, em um domínio de aplicações de objetos distribuídos, fazem parte do projeto *JaCoWeb Security*. Uma descrição detalhada das atividades administrativas para o estabelecimento do esquema de autorização discricionário do *JaCoWeb Security* e do Serviço de Política *PoliCap* pode ser encontrada em [51].

4.2.3- Dinâmica do Controle de Acesso

Como a aplicação não é ciente da segurança, o controle de acesso é realizado de forma transparente à aplicação, no nível do ORB, através dos objetos de serviços descritos na seção 3.9 (***AccesDecision***, ***RequiredRights*** e ***DomainAccessPolicy***).

Conforme apresentado na seção 3.9, a especificação CORBAsec define dois momentos distintos de atuação do interceptador de controle de acesso: o instante *Bind Time*, quando o interceptador obtém as políticas que serão usadas na invocação, e o instante *Access Decision Time* (tempo de decisão de acesso) quando o interceptador, a partir do objeto ***AccessDecision***, verifica se a requisição pode ou não ser atendida. Considera-se então que o objeto ***AccessDecision*** e as políticas que serão usadas na invocação já estão disponíveis no objeto ***Current***.

Na especificação do serviço de segurança CORBA, o objeto ***DomainAccessPolicy*** representa a interface de acesso à política de autorização discricionária. Este objeto representa os **direitos concedidos** a um ou mais principais para execução de operações sobre todos os objetos de um domínio. Para simplificar a administração, o objeto ***DomainAccessPolicy*** pode agregar principais a partir de seus atributos de privilégios (por

exemplo, constituindo grupos ou definindo papéis - *roles*). Se a delegação de privilégios é suportada, podem-se ainda definir direitos diferentes de acordo com o estado de delegação do principal (*initiator*, se este inicia uma invocação, ou *delegate*, se este se encontra em uma cadeia de delegação). Para representar a autorização de um principal (ou um conjunto deles), o CORBAsec define tipos de direitos pertencentes a uma família de direitos. Somente a família CORBA, composta pelos tipos de direitos ***g*** (*get*), ***s*** (*set*), ***m*** (*manage*) e ***u*** (*use*), está descrita na especificação. Entretanto, esta especificação possibilita definir novas famílias de direitos. Um exemplo de um objeto ***DomainAccessPolicy*** é apresentado na Tabela 4.3.

Atributos de Privilégio	**Estado de Delegação**	**Direitos Concedidos (*granted rights*)**
Access_id: alice	Initiator	corba: gs--
group: programadores	Delegate	corba: su--
role:gerente_do_banco	Initiator	corba: gsu-

Tabela 4.3 – Objeto ***DomainAccessPolicy***

Analisando a tabela acima, verifica-se que um usuário que possui o *role gerente_do_banco* possui os direitos *get*, *set* e *use* da família CORBA quando este inicia uma invocação. Pode-se observar ainda que o sujeito do modelo de matriz de acesso tradicional (ver seção 2.3.1) é a combinação de um atributo de privilégio e de um estado de delegação.

No objeto ***RequiredRights***, constata-se que cada operação de uma dada interface possui direitos requeridos ou necessários (atributos de controle do objeto destino) para executar uma invocação a esta operação. Ainda para definir este objeto, existe um combinador de direitos que possibilita especificar se um principal necessita de todos (*All*) os direitos requeridos para executar uma operação, ou se somente a posse de qualquer um (*Any*) dos direitos requeridos são necessários. A representação de um objeto ***RequiredRigths*** é definida na tabela a seguir:

Direitos Requeridos (*RequiredRights*)	Combinador de Direitos	Operação	Interface
Corba:g	All	Ver_saldo	Renda_fixa
Corba:gu	All	Depositar	Conta_bancária
Corba:gs	Any	Depositar	Poupança

Tabela 4.4 – Objeto ***RequiredRights***

Segundo a Tabela 4.4, um usuário necessita dos direitos *set* e *use* para invocar a operação *depositar* da interface *Conta_bancária*.

O objeto ***AccessDecision***, em tempo de decisão de acesso, é o responsável por verificar se uma operação de um objeto destino específico, solicitada por uma aplicação cliente, deve ser permitida ou não. A dinâmica dessa decisão de acesso é ilustrada na Figura 4.3.

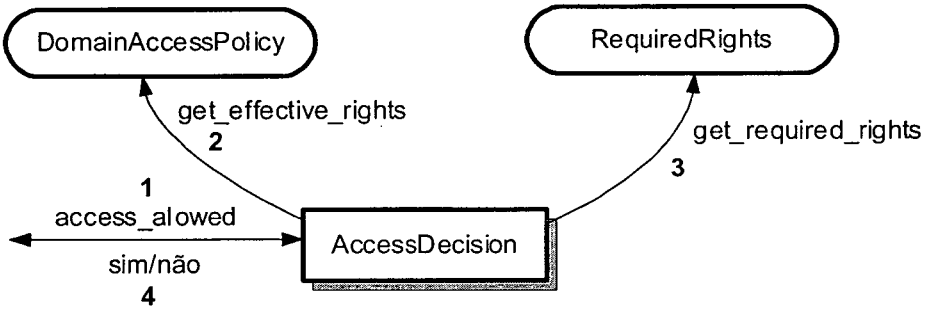


Figura 4.3 – Objeto ***AccessDecision***

As decisões de acesso dependem dos atributos de privilégios do principal, fornecidos pelo objeto ***DomainAccessPolicy***, e de atributos de controles do objeto destino, definidos pelo ***RequiredRights***.

O interceptador de controle de acesso deve, através do método **access_alowed** do objeto ***AccessDecision*** (passo 1 da Figura 4.3), solicitar a verificação de uma invocação de objeto, especificando a operação requerida, as credenciais do principal (que podem ser obtidas pela execução da operação **get_atributes** da interface ***Credentials***), o identificador do objeto destino e o nome da interface do objeto destino. No passo 2, o objeto ***AccessDecision***, de posse dessas informações, solicita ao objeto ***DomainAccessPolicy***, através do método **get_effective_rights**, que informe os direitos concedidos ao principal da invocação, e solicita ao objeto ***RequiredRights***, através do método **get_required_rights**

(passo 3 da Figura 4.3), os direitos necessários para executar a operação da interface especificada. Já no passo 4, o objeto ***AccessDecision*** compara esses direitos requeridos com os direitos efetivos fornecidos pelos objetos de política, e, caso os direitos requeridos foram fornecidos ao objeto de aplicação, o acesso é permitido.

Analisando os exemplos das Tabelas 4.2 e 4.4, tem-se que um *gerente_do_banco*, quando está no estado de delegação *initiator*, possui os direitos *g*, *s* e *u* em um domínio de segurança. Se este principal deseja executar a operação *Depositar* da interface *Poupança*, como todos (*All*) os direitos requeridos - *g* e *s* - são também concedidos a este principal, a operação lhe será permitida.

4.2.4- Estabelecimento e Uso da Associação Segura

O interceptador de chamada segura, apresentado no esquema da Figura 4.1, é o responsável por conduzir o estabelecimento de um contexto de segurança e por usar esse contexto para garantir a integridade e/ou confidencialidade das mensagens entre cliente e servidor. Como descrito na seção 3.7, o interceptador de chamada segura atua em dois momentos distintos: em *Bind Time* ou *binding*, quando a associação segura é estabelecida, e no instante de proteção de mensagem. A dinâmica deste serviço ORB é descrita a seguir.

Em **tempo de *binding***, o interceptador de chamada segura executa os seguintes passos:

1. Define e/ou obtém as configurações necessárias para a invocação no lado do cliente (***QOPPolicy***, ***InvocationCredentialPolicy***, ***SecurityMechanismPolicy***, ***EstablishTrustPolicy***, ***DelegationDirectivePolicy***).

Como a aplicação cliente não é ciente da segurança, serviços ORB irão, transparentemente, aplicar as políticas de segurança. A segurança aplicada é especificada pela:

- política de segurança definida pelo administrador;
- credenciais usadas pelo cliente;
- política de segurança para o objeto alvo (essas informações de segurança estão disponíveis para o cliente na referência do objeto alvo).

A cada primeira invocação sobre um objeto, o ORB estabelece uma associação segura entre o cliente e o objeto alvo, usando para isto a referência do objeto alvo (IOR).

As propriedades da associação segura dependem das políticas que são identificadas na referência de objeto, usando a operação **set_policy_overrides** sobre a referência[4], e dependem das políticas que são definidas como pertencentes ao fluxo de execução da invocação (políticas da *thread*), adicionadas, usando a operação **set_overrides** sobre o objeto ***PolicyCurrent***[5].

2. Analisa as políticas para selecionar um mecanismo, estabelecer a confiança, garantir delegação e selecionar a credencial que deve ser usada na invocação (deve ser compatível com os componentes de segurança da referência do objeto alvo).

As políticas no lado do cliente, que estão definidas para uma referência de objeto particular, empregam uma semântica particular na determinação das características de invocação feitas com a referência de objetos. O ORB executa um **procedimento de decisão** para determinar as características de segurança compatíveis com os mecanismos de segurança que o objeto destino suporta, anexadas na referência do objeto, e as políticas de segurança no lado do cliente.

O **procedimento de decisão** aplicado pelo serviço de segurança ORB (interceptador de chamada segura) determina o mecanismo de segurança, um único objeto **Credentials** compatível e um componente de segurança da IOR do objeto destino para ser usado nas invocações feitas sobre a referência de objeto.

3. Estabelece o contexto de segurança (objetos ***SecurityContext***) que será usado para proteção da requisição (*request*) e das respostas (*responses*).

Utilizando as características de segurança resultantes do **procedimento de decisão,** o interceptador chamada segura do cliente chama o método **init_security_context** do objeto ***Vault*** para iniciar o estabelecimento do contexto de segurança.

---

[4] Esta operação só é executada por aplicações cientes da segurança, e que desejam definir as suas políticas de invocação. Logo, estas políticas não serão consideradas.

[5] Este objeto está sendo incluído na especificação CORBAsec. O **PolicyCurrent** armazena informações de estado associadas aos objetos de política do contexto de execução corrente. Entretanto, como até este momento os mecanismos para sobrescrever políticas não foram ainda padronizados para o **PolicyCurrent**, a exemplo do ORBAsec SL2 [1], as operações **get_overrides e set_overrides** são encontradas na interface *SecLev2::Current*.

O objeto ***Vault*** retorna ao interceptador um *token* de segurança para ser enviado ao objeto destino e indica se haverá continuação de trocas de informações. Este também retorna uma referência para o objeto ***ClientSecurityContext***, recentemente criado para esta associação cliente/servidor.

O interceptador deve construir uma mensagem de estabelecimento de associação, incluindo o *token* de segurança (retornado pelo ***Vault***) que deve ser transferido para permitir o estabelecimento do contexto de segurança do lado do servidor. A mensagem de estabelecimento da associação pode ser construída de duas maneiras:

1- Quando apenas o *token* de segurança inicial é necessário para estabelecer a associação. Após recebida no servidor, a mensagem de estabelecimento da associação é interceptada pelo interceptor de chamada segura do servidor, que no *bind time* chama **Vault::accept_security_context**, para criar o objeto ***ServerSecurityContext***.

2- Quando diversas trocas são requeridas para estabelecer a associação segura, a mensagem de estabelecimento de associação é enviada separadamente. O interceptador envia a mensagem para o objeto destino. Esta mensagem é interceptada no servidor pelo seu interceptador de chamada segura, que invoca, em tempo de ligação, o método **accept_security_context** do objeto ***Vault***, para criar o objeto ***ServerSecurityContext***. Entretanto, neste caso o interceptador do servidor deve gerar um outro *token* de segurança e emiti-lo de volta para o interceptor do cliente O interceptador cliente chama o objeto ***ClientSecurityContext***, através da operação **continue_security_context**.

Quando o interceptador de chamada segura recebe uma mensagem de estabelecimento de associação, este usa o ***Vault*** (se ainda não há contexto de segurança) ou o apropriado objeto contexto de segurança para processar o *token* de segurança. Quando a associação é estabelecida, a função de proteção da mensagem é usada para recuperar o *request* e proteger o *reply*.

Como já mencionado, o objeto ***Vault*** e os objetos de contexto criados por este estão diretamente relacionados com a tecnologia de segurança que será usada para fornecer os controles criptográficos. A especificação do CORBAsec sugere o uso de interfaces de segurança genéricas (GSS-API) para a comunicação entre objetos de serviço e serviços subjacentes, para que assim um objeto ***Vault*** não seja específico a uma única tecnologia de segurança.

Lang [24] afirma que quando um ORB opera sobre uma camada de transporte segura (p.ex., o protocolo SSL), este não precisa fornecer sua própria proteção de

mensagem. A integração do ORB com o SSL não é claramente especificada no CORBAsec. Como não há uma API (*Application Programming Interface*) padrão, geralmente, o ORB tem que manipular diretamente com uma implementação do protocolo SSL (*toolkit* SSL). Em [26], Humenn afirma que o contexto de segurança não precisa ser estabelecido como descrito neste passo, já que os objetos ***Vault*** e ***SecurityContext*** estão baseados no protocolo SECIOP.

Conclui-se nesse trabalho que a integração ORB+SSL não deve ser implementada no núcleo do ORB para que este não perca características importantes como interoperabilidade. Para que essa integração não perca a aplicabilidade em sistemas abertos, propõe-se que o objeto definido como substituível, o ***Vault***, manipule a negociação do contexto de segurança da associação (*handshake* do SSL). Dessa forma, as implementações do núcleo do ORB e dos serviços ORB estarão independentes da tecnologia de segurança subjacente.

4. Torna as credenciais do cliente (incluindo seus atributos de segurança) disponíveis no servidor (colocá-las no atributo *received_credentials* do *Current*).

As credenciais do principal, que estão na lista ***own_credentials*** do objeto ***Current*** do cliente, precisam ser transferidas de forma transparente ao atributo ***received_credentials*** do ***Current*** do objeto alvo. Isto ocorre via mecanismo de segurança subjacente (por exemplo, Kerberos e SSL). Entretanto, a informação transferida não é o objeto ***Credentials***, mas somente a informação que representa este principal. O objeto ***Vault***, no lado do servidor, usando a operação **accept_security_context**, constrói o objeto ***Credentials*** que representa esta informação. Este irá validar essas credenciais, verificando sua confiança em relação à entidade que gerou o certificado para as mesmas. No fim do processo de ligação, essas credenciais estarão contidas no lado do servidor, no atributo ***received_credentials*** do **Current**.

Tomando como exemplo o uso do Kerberos como suporte da tecnologia de segurança subjacente, um cliente precisa que o ***PrincipalAuthenticator*** crie o objeto ***Credentials*** para o Kerberos para que este possa realizar invocações de objetos. Supõe-se que o cliente tenha uma credencial Kerberos com o atributo *AccessId* com o valor de "maria@jacoweb". Este cliente, de posse de sua credencial Kerberos, deseja fazer uma

invocação ao objeto B ("joão@jacoweb"). Para o Kerberos, o objeto ***Vault*** deve obter em um serviço de distribuição de *tickets* um *ticket* para que "maria@jacoweb" possa conversar com "joão@jacoweb". O ***Vault*** usa este *ticket* para gerar um *token* que pode ser entendido somente pelo "joão@jacoweb". Usando o protocolo SECIOP, o objeto ***Vault*** (do cliente) envia este *token* em uma mensagem de estabelecimento de contexto. Se o ***Vault*** (do servidor) entende este *token*, fica fácil verificar quem enviou o *token*. Então, o ***Vault*** do lado do servidor, usando a operação **accept_security_context**, constrói a credencial do cliente (objeto ***ReceivedCredentials***) que contém, basicamente, o atributo *AccessId* com o valor "maria@jacoweb" [25].

Usando o ***SSL*** como tecnologia de segurança, tem-se o CSI nível 0 (*Common Secure Interoperability Level 0*) . Como este nível suporta somente políticas baseadas em identidade sem delegação, basta usar o **DN** (*Domain Name*) do certificado emitido pelo cliente para representar o principal com o atributo ***AccessId***. De acordo com a solução apresentada anteriormente, o objeto ***Vault*** deve, a partir da API da implementação SSL, analisar a cadeia de certificados do cliente para obter o **DN** do principal e construir o objeto **Credentials**, no lado do servidor.

Para utilizar políticas baseadas nos atributos de privilégio definidos no processo de autenticação, a literatura sugere [37], como solução ainda não padronizada pela OMG, o uso de um certificado especial que conterá os atributos de privilégio (PAC- *Privilege Attribute Certificate*). No caso do SSL, que ainda não suporta este certificado, pode-se definir no **DN** (*Domain Name*) um atributo de privilégio (por exemplo, *role*, *level* ou *group*), ao invés do atributo de identidade.

Resumindo, em ***tempo de binding***, o interceptador de chamada segura deve :

- obter as políticas necessárias para a invocação (ver passo 1 da Figura 4.4);
- executar o **procedimento de decisão** (ver passo 2 da Figura 4.4);
- estabelecer o contexto de segurança (objetos SecurityContext) (ver passo 3 da Figura 4.4);
- tornar as credenciais do cliente disponíveis no servidor.

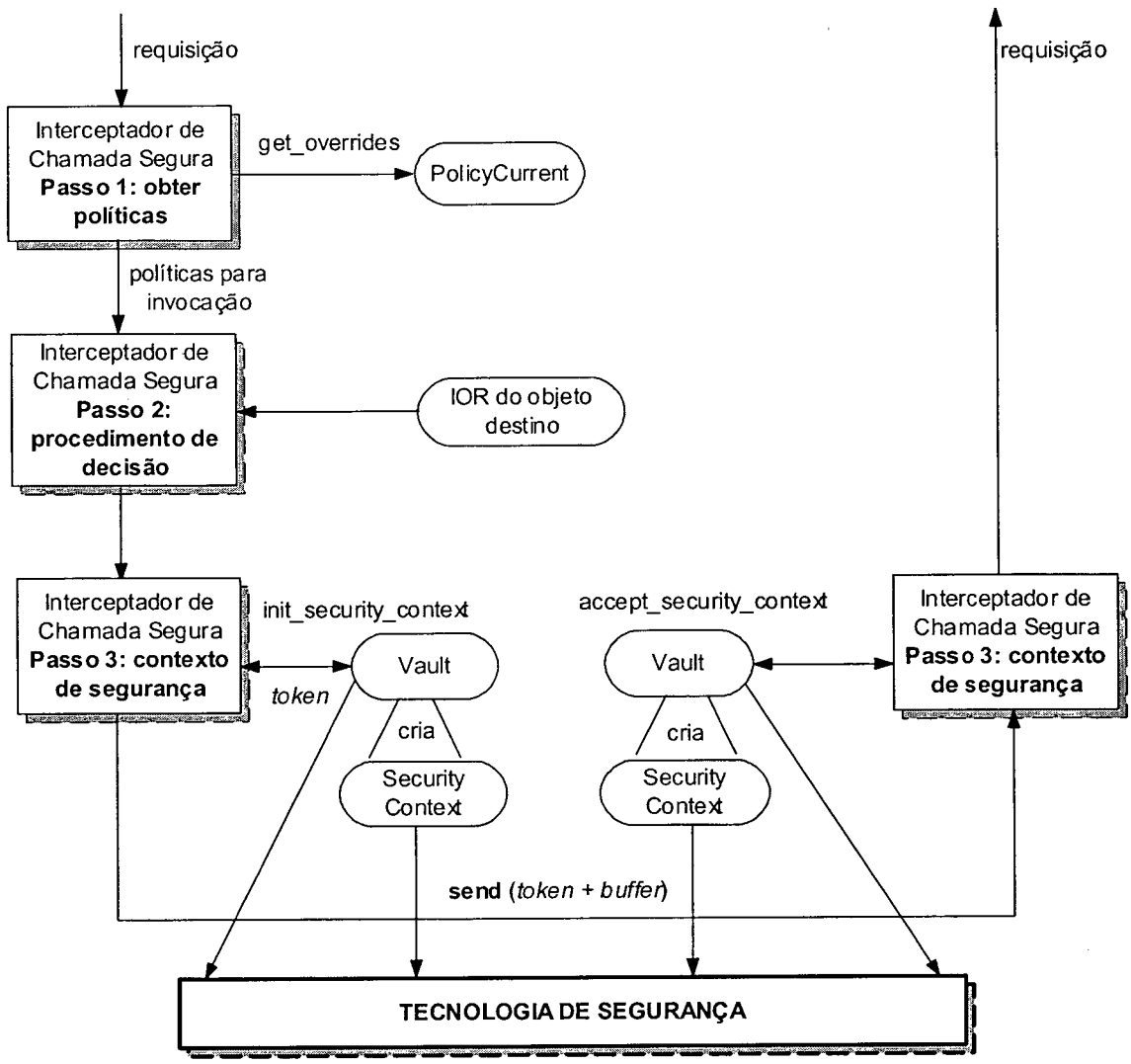


Figura 4.4 – Dinâmica do Interceptador de Chamda Segura durante o *binding*

Depois de estabelecer o contexto de segurança, em **tempo de proteção de mensagem,** o interceptador de chamada segura é usado para proteger as mensagens, fornecendo integridade e/ou confidencialidade para as requisições e respostas.

No lado do cliente, usando o objeto ***ClientSecurityContext***, o método **send_message** do interceptador de chamada segura, chama o método **SecurityContext::protect_message** para proteger as mensagens. Do lado do servidor, o método **receive_message** do interceptador chama o método **reclaim_message** do ***ServerSecurityyContext***, para recuperar a mensagem.

4.3- Modelo do Protótipo Implementado

Após o refinamento da dinâmica do modelo CORBAsec, esta seção introduz o modelo proposto para o protótipo de um esquema de autorização discricionária .

O nível de funcionalidade de segurança que se espera atingir com o esquema de autorização implementado nesta dissertação é o nível 2, cujas interfaces IDLs encontram-se no módulo *SecurityLevel2* do CORBAsec. O gerenciamento das políticas discricionárias de segurança é abordado neste trabalho com base no módulo *SecurityAdim* da especificação CORBAsec. Entretanto, admite-se que, como as aplicações não estarão cientes da segurança que lhes será atribuída, estas não poderão introduzir suas regras para impor suas próprias políticas. Ou seja, as interfaces definidas para a aplicação não serão tratadas no protótipo aqui proposto.

Para a configuração do ORB seguro no *SecurityLevel2* é necessário definir um conjunto de propriedades, que poderá ser usado para configurar os protocolos de segurança, os mecanismos e outras características da inicialização do canal seguro. Caso este conjunto de propriedades não seja definido, devem-se usar configurações previamente definidas como padrão.

A Figura 4.5 representa o esquema de autorização simplificado que foi implementado nesta dissertação. Após a inicialização do ORB e do POA (somente para o servidor), deve ocorrer a inicialização da estrutura de segurança do modelo proposto, definida pela obtenção do objeto ***Current.*** Para finalizar essa inicialização, a classe responsável pela inicialização do ORB seguro fornece uma coleção de métodos, definidos estaticamente, que são necessários para a criação de objetos que representam as configurações da associação segura a ser estabelecida.

Conforme apresentado na Figura 4.5, a seguir, o protocolo SSL (*Secure Socket Layer*) foi empregado como tecnologia de segurança subjacente. Dentre as várias tecnologias de segurança em sistemas distribuídos, o SSL, desenvolvido pela *Netscape*, se destaca por ser um protocolo criptográfico amplamente utilizado em aplicações na Internet. Este protocolo é utilizado neste trabalho por estar inserido pela OMG, em sua última revisão do serviço de segurança, como um dos protocolos a serem utilizados por aplicações que implementam a segurança no CORBA. A escolha do SSL para o estabelecimento da

associação segura garante ao protótipo o nível de interoperabilidade 0 (CSI nível 0) e políticas de autorização baseadas em identidade.

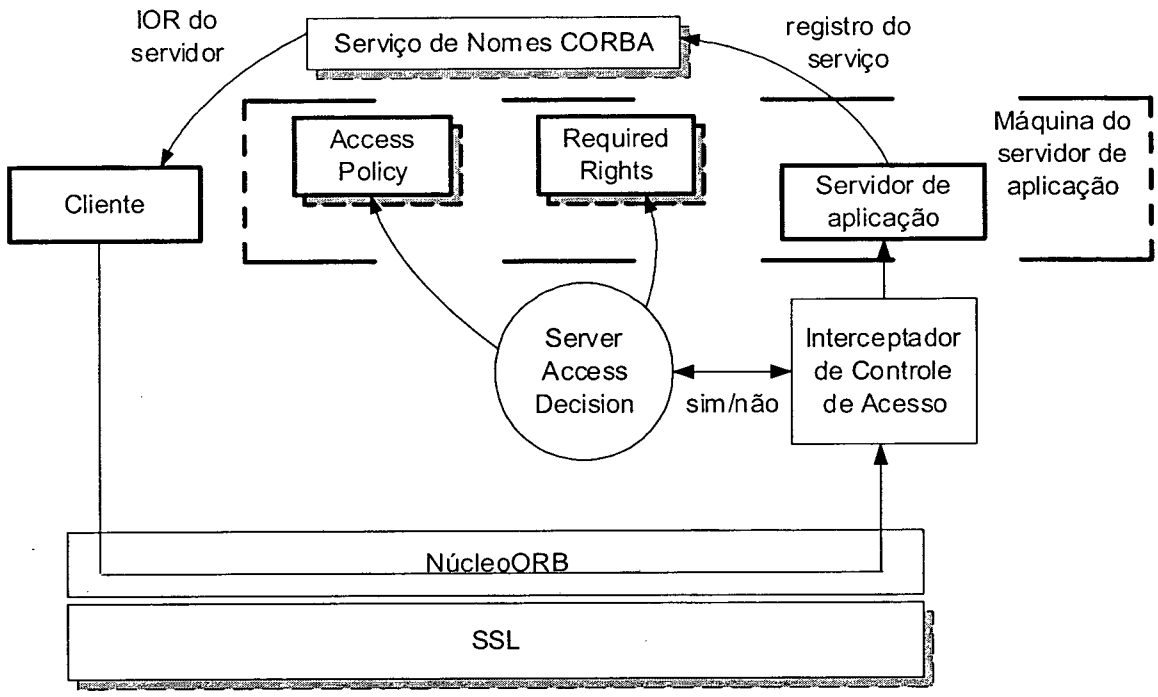


Figura 4.5- Esquema de Autorização Implementado

O interceptador de controle de acesso da Figura 4.5 tem como função aplicar a política de autorização discricionária definida nos objetos ***AccessPolicy*** e ***RequiredRights***. Estes objetos, residindo na mesma máquina do servidor da aplicação, torna desnecessário o uso de *capabilities* em um segundo nível de controle. A inserção, atualização e remoção de direitos nesses dois objetos são executados pelos próprios objetos servidores da aplicação logo depois de se registrarem no serviço de nomes CORBA.

A implementação do protótipo introduzido nesta seção está detalhada no Capítulo 5.

4.4- Conclusões do Capítulo

O esquema de autorização discricionária descrito neste capítulo é parte do projeto *JaCoWeb Security* que visa comprovar a aplicabilidade do modelo CORBAsec em conjunto com outras tecnologias, tais como Web e Java, em sistemas de larga escala.

Não se tem conhecimento de projetos, como o *JaCoWeb Security*, que abordem o uso do serviço de segurança CORBA para aplicações de larga escala, desenvolvidos em universidades brasileiras. Com o refinamento das abstrações da especificação CORBAsec, pode-se comprovar que muitas soluções de ORBs seguros não estão baseadas no uso do serviço de segurança do CORBA especificado pela OMG. Acredita-se que as abstrações do modelo CORBAsec em conjunto com problemas como, interoperabilidade entre ORBs seguros, gerência de domínios de segurança, gerenciamento de políticas de autorização e de controles criptográficos, dificultam a aplicabilidade deste modelo em aplicações reais.

Capítulo 5: Implementação

5.1- Introdução

No capítulo 4 foi apresentado um protótipo de um esquema simplificado de autorização discricionária baseado nas abstrações do serviço de segurança CORBA para aplicações distribuídas em redes de larga escala. De forma a verificar a viabilidade e a confiabilidade de um esquema de autorização, este capítulo descreve assim a implementação de um protótipo que visa fornecer segurança a um sistema *Internet Banking* fictício.

Como um exemplo de rede de larga escala, a Internet vem despontando como o grande veículo dos tempos modernos [42]. Vantagens como, abrangência global, tecnologia difundida e custo de comunicação reduzido, tem despertado o interesse de diversos setores da economia. No mundo, há cerca de 320 milhões de usuários da Internet, já no Brasil os números se aproximam a 4,9 milhões [42]. Entretanto, a Internet é uma rede de duas mãos, pois assim como ela pode tornar disponível informações de milhões de usuários, ela também possibilita que invasores violem a segurança de muitos sistemas [13].

O ambiente *World Wide Web* (WWW) foi criado para facilitar o acesso a grandes quantidades de informações existentes na Internet. Este ambiente vem sendo usado por corporações para distribuir informações e conduzir transações de negócios. Contudo, *browsers* e servidores *Web* não estão sendo empregados de forma mais ampla, devido ao fato da segurança dos acessos via ambiente *Web* ainda ser limitada, especialmente em termos de autenticação e autorização do acesso de informações.

O comércio eletrônico ainda está iniciando a sua expansão no Brasil, representando 15 % das transações eletrônicas realizadas. A principal aplicação, com 27 % de uso, é o *Home/Internet Banking* [27]. Os serviços de *Home/Internet Banking* são utilizados por 51% dos brasileiros, que "navegam" na Internet [43].

De acordo com os últimos congressos sobre automação bancária, o uso de serviços de *Internet Banking* vem surgindo como solução para a crescente necessidade de serviços mais eficazes [43]. O reduzido custo das transações financeiras e uma melhoria do serviço, como a disponibilidade do serviço 24 horas por dia, o *Internet Banking*, vem agradando as instituições financeiras e seus clientes. Apesar de ser uma grande tendência, somente cerca de 50 % dos bancos brasileiros oferecem este serviço.

Por ser uma aplicação crítica amplamente utilizada em ambientes de larga escala e pelas características e necessidades apresentadas acima, escolheu-se um sistema de *Internet Banking* para avaliar a potencialidade do esquema de autorização discricionário, proposto neste trabalho.

Este capítulo tem por objetivo descrever o protótipo implementado, apresentando as ferramentas utilizadas, discutindo as dificuldades encontradas para cumprir o modelo do *CORBAsec* e a viabilidade de um esquema de autorização discricionária.

5.2- Estrutura Geral do Protótipo

O exemplo adotado é composto por um objeto servidor CORBA, um *applet* cliente Java e uma aplicação *stand*-alone cliente responsável por algumas funcionalidades administrativas do sistema bancário. A Figura 5.1 ilustra a estrutura distribuída do protótipo implementado.

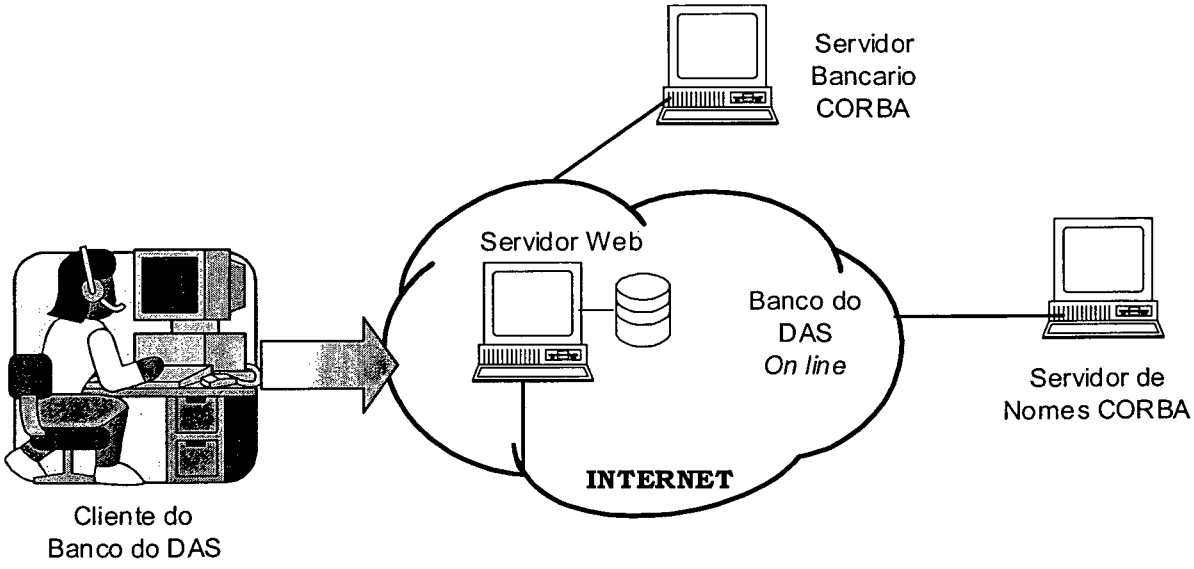


Figura 5.1 – Estrutura do Protótipo Implementado

Analisando a Figura 5.1 observar-se que, de qualquer ponto da rede Internet, um cliente do **Banco do DAS** pode, com o auxílio de um navegador, se comunicar com o seu servidor bancário. Um cliente deve fornecer o endereço da página *Web* do **Banco do DAS**, para que a carga automática do *applet* seja realizada na estação cliente. Quando uma operação é solicitada, o *applet* cliente se comunica com um serviço de nomes CORBA, para obter a referência do objeto servidor bancário. De posse dessa referência, e se for um usuário autorizado, o cliente estabelece uma associação segura com o objeto CORBA servidor. Nesta implementação, o servidor Web que armazena o código do *applet* cliente, o objeto CORBA (servidor bancário) e o serviço de nomes CORBA não necessitam estar na mesma máquina.

O objeto servidor CORBA foi desenvolvido com a ferramenta JacORB 1.0 [5] e o *applet* cliente e a aplicação *stand_alone* foram implementados 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.

Conforme introduzido no capítulo anterior, para o estabelecimento da associação segura foi empregado como tecnologia de segurança subjacente, o protocolo SSL.

5.2.1- O Protocolo SSL e o pacote iSaSiLk

O SSL [12] é um protocolo de propósito geral adequado para proteger conexões em sistemas distribuídos, prover autenticação, confidencialidade e integridade para comunicações através de conexões TCP/IP. Este protocolo está localizado entre o protocolo da camada de transporte, o TCP e o protocolo da camada de aplicação, e é composto por duas camadas: o protocolo ***SSL Record***, que provê conexões seguras garantindo confidencialidade e integridade das mensagens transportadas, através de criptografia simétrica e de mensagens de integridade (MAC – *message aunthentication code*); e o protocolo ***SSL Handshake***, que permite a autenticação de cliente e servidor, negocia algoritmos criptográficos e gera a chave simétrica usada pela camada *SSL Record* (ver Figura 5.2).

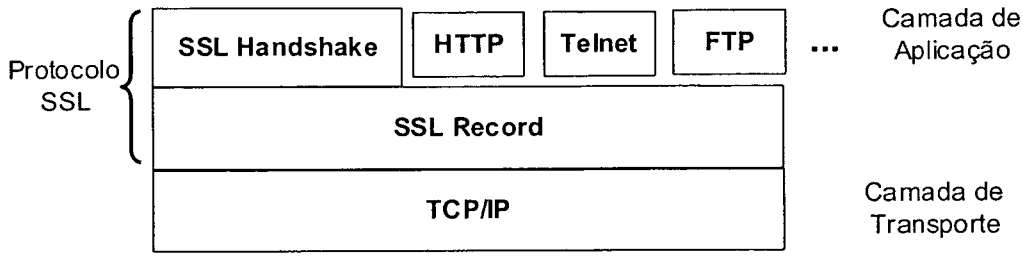
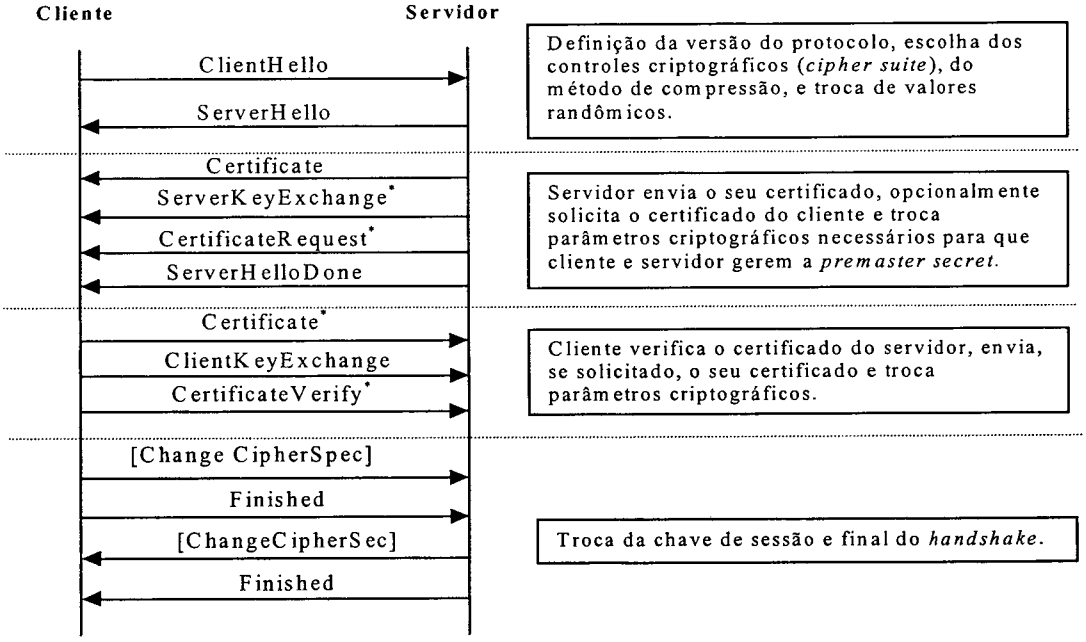


Figura 5.2- Arquitetura SSL

O protocolo **SSL *Record*** é usado para transferir dados de aplicação e de controle SSL entre cliente e servidor. Estes dados estão possivelmente fragmentados em pequenas unidades, e podem ser comprimidos, assinados e cifrados antes de serem transmitidos pelo protocolo de transporte confiável (TCP).

O SSL utiliza, para autenticação de cliente e servidor, certificados digitais e transferências com assinaturas digitais. A sessão SSL é estabelecida quando a negociação inicial (*handshake*) entre cliente e servidor é realizada. O fluxo de mensagens durante o *handshake* está representado na Figura 5.3.



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

Figura 5.3- Fluxo de mensagens do *handshake*.

A autenticação no SSL usa o X.509 *certificates* [19] para transferir informações sobre a chave pública de uma aplicação. O SSL v3 usa o *X.509 v3 certificates* para assegurar chaves RSA, e um modificado *X.509 certificates* para assegurar chaves públicas,

usadas pelo protocolo de distribuição de chaves do Departamento de Defesa dos EUA, *Fortezza/DMS*.

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

Utilizou-se neste trabalho o pacote *iSaSiLk* que implementa a versão atual do protocolo SSL (SSL v3), através da linguagem de programação Java. Este pacote foi desenvolvido no *Institute for Applied Information Processing and Communications* da Universidade Tecnológica de Graz, na Áustria, e pode ser obtido sem custo, para fins não comerciais, em http://jcewww.iaik.at/iSaSiLk. Escolheu-se o pacote iSaSiLk [17] por apresentar todas as funcionalidades necessárias para o protótipo proposto.

5.2.2- O *Applet* Cliente e o pacote JDK 1.2.1

A linguagem Java é considerada um padrão de *facto* para a programação de aplicações distribuídas. Essa linguagem popularizou o conceito de códigos móveis através de seus *applets* executados a partir de *browsers* Web, e hoje é conhecida como a *linguagem da Internet* [49]. Essa mobilidade de código possibilita a um servidor enviar códigos para os clientes para que eles sejam executados localmente, o que alivia a carga de processamento deste servidor.

Applets clientes estão sujeitos às restrições de segurança impostas pelo *browsers* onde são executados. Estas restrições impedem que esses códigos possam ter acesso ao disco local ou façam conexões com sítios (*hosts*) diferentes dos que eles foram carregados. De alguma maneira isto impõe que os objetos servidores da aplicação estejam na mesma máquina do servidor Web, a partir do qual o *applet* foi carregado. Algumas técnicas são oferecidas para permitir que os *applets* se libertem dessas restrições e possam se comunicar com objetos em um sistema distribuído sem limitação de localização. Neste trabalho, adotou-se como solução o uso de um *applet* assinado segundo a API JCA (*Java Cryptography Architecture*) [45] da plataforma Java™. Na implementação do *applet* cliente, bem como no processo de assinatura e verificação do *applet*[1], utilizou-se a versão 1.2.1 do *toolkit* de desenvolvimento Java (*Java Development Kit* – JDK).

[1] Um relatório do processo de assinatura do *applet* cliente está disponível no endereço http://www.lcmi.ufsc/jacoweb/documentos.

A assinatura digital foi utilizada para permitir que o *applet* cliente implementado possa estabelecer conexões com um serviço de nomes CORBA e com o servidor bancário, localizados em máquinas diferentes do servidor Web.

Para fornecer as funcionalidades de criptografia da API JCE (*Java Criptography Extension*), que possui restrições de exportação[2], utilizou-se uma implementação deste pacote, desenvolvido pela Universidade Tecnológica de Graz, na Áustria (*toolkit* JCE-IAIK).

5.2.3- JacORB

Toda a implementação do protótipo teve como plataforma de desenvolvimento o JacORB beta 13 [6], protótipo desenvolvido na Universidade de Berlim. Esse *middleware* corresponde a um conjunto de suportes e ferramentas de desenvolvimento que permitem construir e integrar aplicações orientadas a objetos em sistemas abertos. O JacORB é completamente desenvolvido em Java, trazendo as vantagens desta linguagem, seguindo as especificações da OMG. Todos os componentes do JacORB se comunicam usando o protocolo IIOP, também padronizado pela OMG. Além disso, o JacORB, dentre os vários serviços que possui, apresenta como destaque o suporte *multithread*, o POA (*Portable Object Adapter*), *gateway* para comunicação dos *applets* e interceptadores.

Quando um cliente faz uma invocação de método em um objeto servidor remoto, há todo um conjunto de procedimentos no tratamento da requisição para que esta possa ser transmitida no canal de comunicação até chegar ao servidor. Uma requisição do cliente é transformada em um objeto ***Request***, o qual contém um conjunto de atributos que define o objeto destino (o servidor), a operação requisitada (o método), os argumentos da invocação, a resposta, entre outros; e um conjunto de métodos que permitem manipular esses atributos. Uma vez criado pelo ORB, a partir de uma requisição do cliente, o objeto ***Request*** passa pelo processo de conversão (*marshalling*) para *byte stream* para que possa ser enviado pela camada de transporte.

Na figura 5.4, é apresentado o modelo de implementação do ORB/CORBA do projeto JacORB . Em detalhe, é apresentado o fluxo de uma invocação dentro do ORB, a partir do ponto em que os dados da requisição (em *byte stream*) alcançam o *socket* no lado

[2] A regulamentação federal corrente nos Estados Unidos da América prevê restrições para a exportação de controles criptográficos fortes.

servidor. O módulo *BasicAdapter*, através do objeto *Connection*, trata das funções de baixo nível que manipulam as conexões TCP/IP entre clientes e servidor (os objetos *ServerSocket*). É importante ressaltar que é criada apenas uma conexão para cada par cliente/servidor, não importando quantos objetos no servidor estão sendo utilizados pelo cliente. O objeto *RequestReceptor* faz a decodificação (*unmarshalling*) do conjunto de *byte stream* para re-montar o objeto *Request* no lado servidor. Então, o *Request* é transformado em *ServerRequest* para ser mandado para a fila de requisições (*RequestQueue*), onde é ordenado em seqüência para chegar ao POA (*Portable Object Adaptor*).

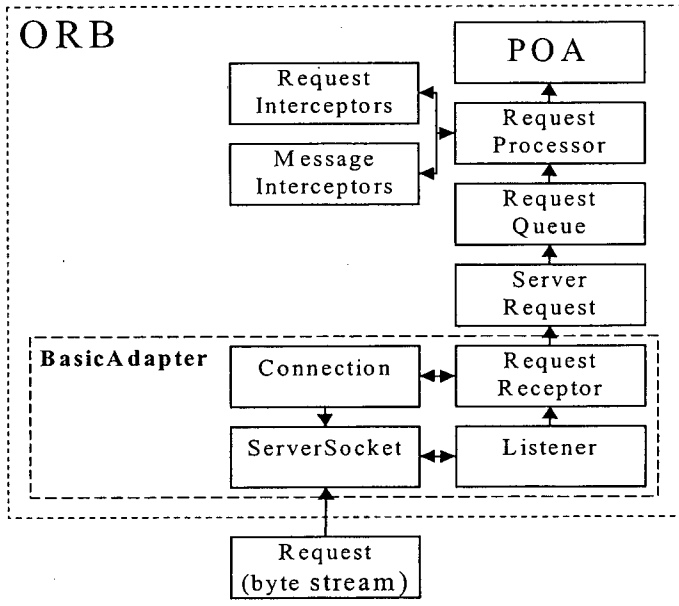


Figura 5.4. Fluxo de dados no JacORB.

Por definição, o POA adapta o conceito da linguagem de programação utilizada para implementar um *servant* ao conceito de objeto CORBA [1]. Um *servant* é uma entidade de uma linguagem de programação particular (por exemplo, um objeto em Java ou uma coleção de funções em C). Um objeto CORBA é uma representação remota de um objeto *servant* diante do ORB – e permite, deste modo, que o *servant* possa ser identificado, localizado e endereçado por uma referência de objeto (IOR). As principais funções do POA são geração e interpretação de referência de objeto, invocação de métodos, ativação, desativação de objeto e implementações, mapeamento de referência de objeto para implementação, etc.

5.3- Modelo Implementado

Baseado no esquema de autorização discricionária utilizado nesta dissertação, um protótipo de um sistema bancário que utiliza um ORB seguro (conforme o CORBAsec) como canal de comunicação foi implementado como descrito a seguir.

5.3.1- Inicialização do ORB seguro

Em [1] tem-se a definição de algumas propriedades que podem ser usadas pelo programador da aplicação para a inicialização de um ORB seguro. Com base nessas propriedades definiu-se, para o projeto *JaCoWeb Security*, a propriedade **jacoweb.seciop** usada para especificar onde o protocolo SECIOP deve ser habilitado (cliente e/ou servidor) e as propriedades **jacoweb.seciop.host** e **jacoweb.seciop.port** que especificam um sítio (*host*) e uma porta de comunicação para conexões SECIOP, respectivamente. Para definir o uso do protocolo SSL, tem-se a propriedade **jacoweb.ssliop**. E, para usar comunicações CORBA padrão (não seguras), definiu-se a propriedade **jacoweb.iiop** que especifica onde o protocolo IIOP deve ser habilitado.

5.3.2- Credenciais e Certificados

As credenciais usadas neste experimento são formadas pelo atributo de privilégio ***role*** da família CORBA. O valor deste atributo (*value*) e a autoridade que o valida (*defining_authority*) são obtidos a partir dos certificados[3] do principal. Os campos *value* e *defining_authority* são obtidos a partir do campo *subjectDN* e *issuerDN* do certificado, respectivamente. Conforme sugerido na especificação do CORBAsec, estes campos são codificados através da classe **Opaque**, que implementa um simples algoritmo para cifragem dos dados.

O objeto ***PrincipalAuthenticator***, responsável pela autenticação e criação do objeto ***Credential***, não foi implementado nesse protótipo. As credenciais do protótipo são criadas estaticamente (a partir dos certificados) e possuem atributos de segurança que identificam os direitos do cliente no sistema (atributo de privilégios).

Os certificados usados nesta implementação foram criados utilizando como ferramenta o pacote JCE-IAIK. Todos os clientes do sistema bancário, usuários do *internet*

[3] Um certificado é formado pelas seguintes informações: sujeito do certificado (*subjectDN*); o emissor do certificado (*issuerDN*); período de validade; informações administrativas (versão e número de série) e informações extras.

banking (*clienteWeb*) ou gerentes, devem possuir certificados, tendo como emissor o *Banco do DAS*. Para o campo *subjectDN* é definido um *role*, que pode ser *clienteWeb* ou *gerente*, representando o atributo de privilégio do cliente. Para o *Banco do DAS* foi criado um certificado auto-assinado[4].

5.3.3- Controle de Acesso Discricionário

Este experimento implementa políticas discricionárias baseadas em ACL's, conforme descrito na seção 4.2.4, estando limitado a uma única verificação de controle de acesso (no lado do servidor). O protótipo implementado está limitado ainda a um único domínio de nomes. A Figura 5.5 sintetiza o modelo implementado para realização do controle de acesso.

O objeto ***RequiredRights***, que contém os direitos requeridos para executar uma determinada operação, é criado estaticamente com os direitos definidos para cada operação da IDL do servidor bancário, conforme apresentado na Tabela 5.3.

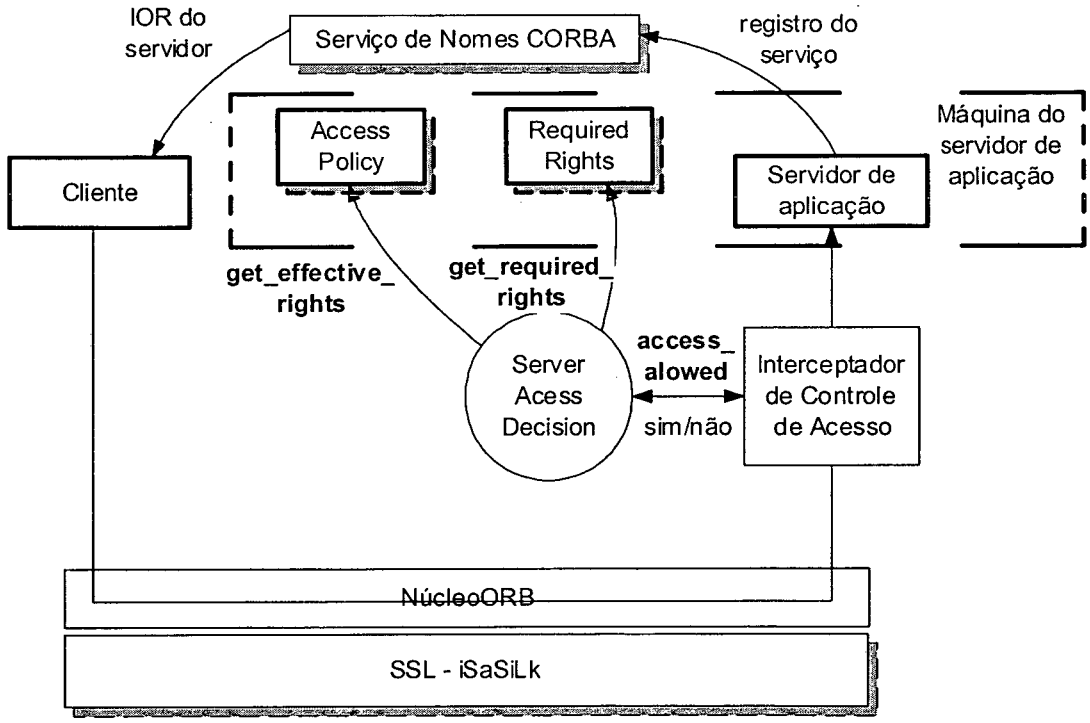


Figura 5.5- Prótipo Implementado para o Controle de Acesso

Descrição	Rights

[4] Em um certificado auto-assinado o emissor e o sujeito são os mesmos.

Registrar novas contas	Jacoweb:**rc**
Apagar conta	Jacoweb:**ac**
Verificar a existência de uma conta	Jacoweb:**ec**
Obter o número de contas do banco	Jacoweb:**nc**
Obter informações de uma dada conta	Jacoweb:**oc**
Efetuar um depósito	Jacoweb:**d**
Efetuar um saque	Jacoweb:**s**
Verificar o saldo	Jacoweb:**vs**
Retirar um extrato	Jacoweb:**e**
Retirar um extrato datado	Jacoweb:**ed**
Efetuar uma transferência	Jacoweb:**t**
Trocar a senha	Jacoweb:**ts**

Tabela 5.3- Direitos definidos para família Jacoweb

O objeto ***AccessPolicy*** é construído dinamicamente a partir dos direitos atribuídos ao cliente (*gerente* ou *clienteWeb*) por meio de um aplicativo Java (aplicativo *decisão de acesso*). Os direitos podem ser definidos de acordo com as opções apresentadas pelo aplicativo *decisão de acesso*, ilustrado na Figura 5.6. Na Tabela 5.4, tem-se um exemplo de um objeto ***AccessPolicy*** obtido a partir deste aplicativo.

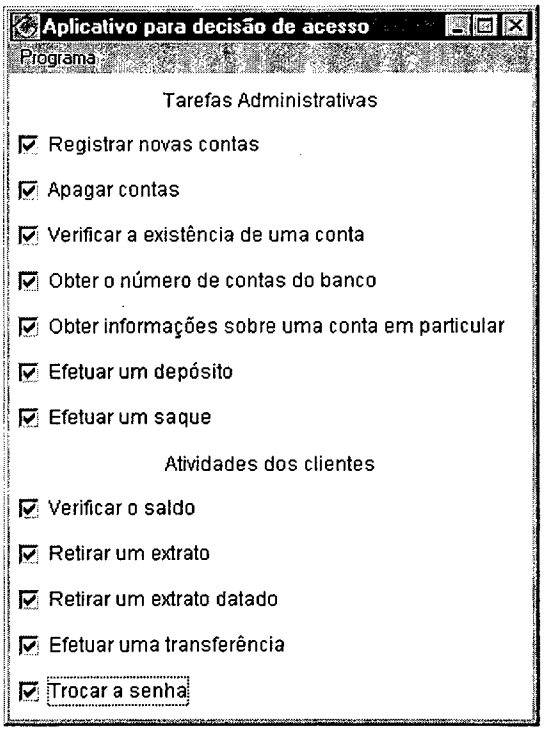


Figura 5.6: Tela do Aplicativo *Decisão de Acesso*

Role	Direitos Efetivos
Gerente	"rc", "ac", "ec", "nc", "oc", "d", "s"
Cliente Web	"vs", "e", "ed", "t", "ts"

Tabela 5.4 – Exemplo de um Objeto AccessPolicy

Em tempo de decisão de acesso, o interceptador de controle de acesso invoca o objeto ***AccessDecision***, através do método **access_alowed**, para que este verifique se o principal tem direitos para requerer a operação. A implementação da operação **access_alowed**, obtém os direitos requeridos para executar a operação especificada, através do método **RequiredRights::get_required_rights**, e comparar com os direitos efetivos obtidos através do método **AccessPolicy::get_effective_rigths**. Caso o principal não tenha direito de acesso, a exceção *PermissãoNegada* é levantada.

5.3.4- Integração ORB + SSL

A solução para integrar o SSL ao ORB, descrita na seção 4.2.5 sugere o uso do objeto ***Vault*** para manipular o pacote que implementa o protocolo SSL. Como o pacote *iSaSiLk* não fornece interfaces que possibilitariam ao ***Vault*** realizar a negociação do contexto de segurança, essa solução não pôde ser implementada.

O *framework* para a integração do ORB/CORBA com suporte SSL consistiu então em encapsular um pacote SSL pronto (o pacote iSaSiLk) na forma de objeto de serviço, tal como os objetos COSS. Na Figura 5.7, são apresentados os objetos, com interfaces definidas em IDL, que compõem o pacote *JaCoWeb Security* para a integração ORB+SSL. A classe ***Security*** encapsula as funcionalidades do pacote iSaSiLk e é instanciada pela aplicação no momento de sua criação. O conjunto de parâmetros definidos na criação do objeto ***Security*** permite ao iSaSiLk definir o **contexto SSL**. O **contexto SSL** define como cliente e servidor devem selecionar de forma compatível os controles criptográficos (*cipher suite*). A seleção dos controles criptográficos é definida pelos seguintes componentes: método de distribuição de chaves - certificados (*key exchange algorithm*), cifragem para transferências de dados (*symmetric encryption algorithm*) e *message digest* para criação do código de autenticação de mensagem (*hash algorithm*).

Na Figura 5.7, têm-se os objetos ***ServerSocket*** e ***SSLServerSocket***. Um para estabelecer conexões normais através do ***Listener*** e outro para estabelecer conexões

seguras através do ***SSLListener***. O ***SSLServerSocket*** é fornecido pelo iSaSiLk e é encapsulado pelo pacote *JaCoWeb*. O ***SSLConnection*** tem a função de instanciar o ***SSLSocket*** para o estabelecimento de uma conexão segura entre cliente e servidor. Portanto, o estabelecimento da chamada segura é realizado dentro do pacote *JaCoWeb* e, como resultado, o conjunto de *byte stream* da requisição cliente é passado para o ORB através do ***SSLRequestReceptor***. Convém notar que ambos ***Listener*** encaminham o conjunto de *byte stream* para o seu respectivo objeto receptador de requisição (***RequestReceptor*** e ***SSLRequestReceptor***, respectivamente) que realiza a conversão para remontar o objeto ***Request***. A partir desse momento, a requisição pode seguir seu caminho normal até o POA, onde ocorre a transformação dos parâmetros do objeto ***Request*** em argumentos a serem passados ao *servant*.

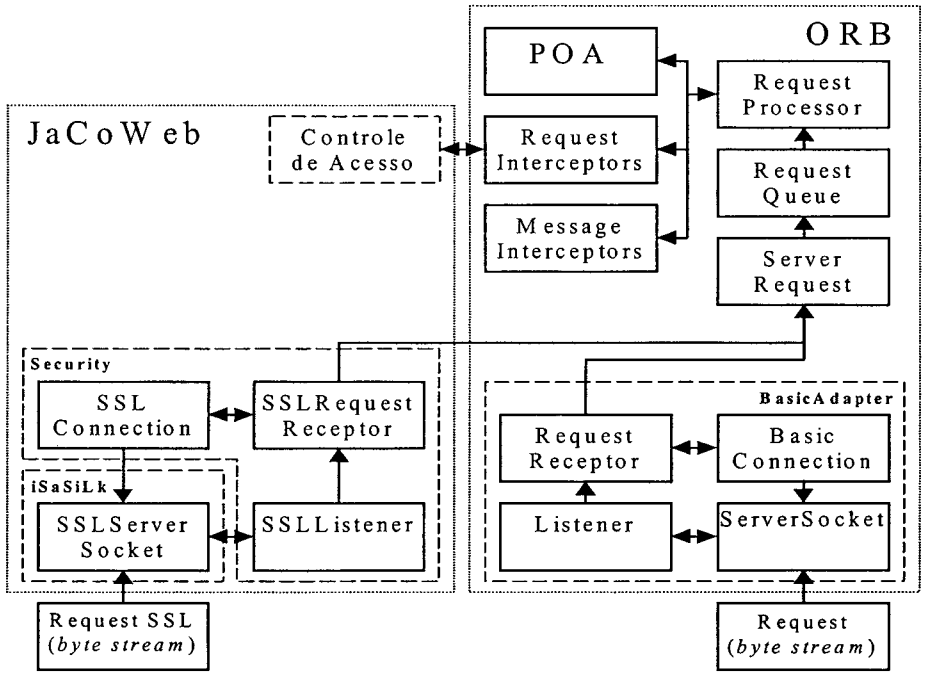


Figura 5.7- O *framework* ORB+SSL.

Neste experimento, o interceptador de chamada segura deixa de ter uma função mais ativa no *framework*, e é utilizado somente para obter informações sobre o contexto de segurança do objeto SSL. No entanto, pode também ser utilizado para fins relacionados com a aplicação.

Vale ressaltar que, apesar do servidor ter disponível duas portas de acesso (uma segura e outra convencional), qualquer tentativa indevida de acesso a um serviço (objeto)

com restrições de segurança pela porta convencional é inibida pelo serviço de controle de acesso no interceptador de requisição (Figura 5.7).

O ponto chave para esta integração dar certo é definir como a aplicação pode selecionar o modo de conexão a ser utilizado (segura ou convencional). No modelo CORBA, um objeto pode ser localizado nos sistemas distribuídos através da sua referência de objeto. Na OMG, a referência de objeto é definida numa forma padrão conhecida como IOR (*Interoperable Object Reference*). A IOR [2] é uma seqüência de caracteres que, quando convertida para o formato adequado, fornece as informações de endereço IP do sítio, da porta, de um ponteiro para o objeto a ser acessado e algumas informações de controle relacionadas à própria IOR. Cada objeto distribuído deve ter sua IOR disponível, registrada em um serviço de nomes. A especificação CORBAsec define uma extensão da IOR de forma que possa agregar informações especificas para que o ORB trate também de conexões seguras. A extensão da IOR é feita adicionando um novo tipo de *Tag* (etiqueta) de segurança ao corpo do IIOP *profile* da IOR.

Quando as aplicações desejam estabelecer associações seguras utilizando o *framework* ORB+SSL, algumas informações de segurança precisam ser inicializadas. O objeto ***Security*** é o responsável pela iniciação da segurança, conforme definido em sua interface IDL (ver Figura 5.8).

Como descrito no esquema de autorização discricionário (ver seção 4.3.1), após a inicialização do ORB e do POA (somente para servidores), as aplicações devem completar a inicialização do ORB seguro. Nesta implementação deve-se instanciar um objeto ***Security***, de acordo com o seu tipo de aplicação (cliente *stand-alone*, *applet* ou servidor).

Para o *applet* cliente, as informações de segurança são definidas através do primeiro método da interface IDL, apresentadas na Figura 5.8 (**initAsClient (orb,applet)**). Este método instancia um objeto ***Security*** e invoca o método **initAsClient (orb)**. Já para aplicações *standalone*, o contexto de segurança pode ser definido através do método **initAsClient (orb)**, que deve: a partir do certificado do cliente definir o principal; criar a credencial SSL estática do cliente (com base no certificado deste cliente); criar e definir os objetos de política utilizados no estabelecimento da associação segura; definir o contexto SSL do cliente, especificando os *cipher suites*, o objeto ***TrustDecider*** (responsável pela

autenticação do servidor) e o certificado do cliente; e finalmente definir estas configurações de segurança (objeto ***Security***) na classe **jacorb.orb.Environment**.

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

Figura 5.8. Interface IDL do pacote JaCoWeb

A Figura 5.9 apresenta a *Tag* com definições para conexões SSL. Basicamente, a estrutura de dados associada com o componente SSL encontra-se definida no módulo SSLIOP [3]. O aumento do tamanho da IOR não traz nenhum problema de compatibilidade adicional, uma vez que a estrutura da IOR é bastante flexível e o próprio serviço de nomes já suporta IORs de tamanhos variados.

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

Onde:

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

Figura 5.9. Estrutura do *Tag* com definições para conexões SSL.

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

5.4- Apresentação dos Resultados

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 apresenta vantagens como transparência da segurança para aplicações e maior confiança nos serviços de segurança que são sempre executados.

Como a especificação CORBAsec não define a forma como o ORB deve manipular a camada SSL, encontraram-se dificuldades na integração ORB+SSL. A negociação inicial (*handshake*) e o uso do SSL, no protótipo, são executados transparentemente pelo iSaSiLk (implementação do protocolo SSL escolhido).

A falta de uma definição, por parte da OMG, de como o SSL deve ser integrado ao ORB de forma a garantir segurança a aplicações distribuídas, limita algumas características desses sistemas abertos, como por exemplo a interoperabilidade e a portabilidade. Acredita-se que se houvesse uma GSS-API definida para manusear o protocolo SSL, e se o objeto ***Vault***, a partir desta API, pudesse invocar o SSL para que este estabelecesse a associação segura, a interoperabilidade seria mais facilmente garantida[5]. Como as interfaces oferecidas aos programadores de aplicações do pacote iSaSiLk não permitem que o objeto ***Vault*** tenha o controle sobre o *handshake* SSL, não foi possível realizar teste da manipulação do SSL a partir do objeto ***Vault***. A seção, a seguir, contém alguns resultados relativos ao desempenho do uso do *framework* ORB+SSL.

[5] O objeto ***Vault*** não precisa de uma API específica para o SSL já que este pode ser exclusivamente dependente de uma tecnologia de segurança específica. Entretanto, o uso de uma GSS-API para manipular o SSL aumentaria a flexibilidade do objeto ***Vault*** para suportar diferentes tecnologias.

Conforme apresentado na seção 3.12, o CORBAsec não especifica os procedimentos padrões para gerenciar os membros dos domínios de políticas de segurança. De forma a simplificar esta limitação, considerou-se somente um domínio para operar com o protótipo *internet banking*

A implementação do controle de acesso discricionário, realizada no lado do servidor, foi facilitada devido ao fato de os objetos da política de segurança discricionária e de o servidor bancário estarem na mesma máquina. Um melhoramento deste controle de acesso fornecendo dois níveis de proteção, através de um servidor de autorização (no lado do clinte) e de uma verificação de *capabilities* (no lado do servidor), é o próximo passo que o projeto *JaCoWeb Security* pretende dar em direção à implementação do esquema de autorização proposto em [50].

5.4.1- Desempenho da integração ORB+SSL

Neste item são apresentadas as análises de desempenho do *JaCoWeb Security*, realizadas com o sentido de avaliar a implementação. As medidas foram realizadas com um servidor e um cliente. O objetivo dessas medidas é avaliar o custo adicional ao integrar mecanismos de segurança (SSL) ao ORB. A fim de possibilitar essa análise, foram comparadas as medidas realizadas em um ORB convencional (JacORB) com o *JaCoWeb Security*.

A primeira medida realizada foi a da latência. No *JaCoWeb*, quando o cliente faz a primeira requisição no servidor, é executado um conjunto de procedimentos para o estabelecimento do contexto de segurança SSL – *handshake* SSL (Figura 5.3). O algoritmo de chave pública, usado para autenticação mútua e distribuição de chaves, foi o RSA de 1024 bits, tamanho utilizado na maioria das organizações. Para cifragem dos dados, utilizaram-se o algoritmo simétrico de cifragem em bloco 3DES_EDE e a função *hash* de segurança SHA para a computação do MAC.

A Figura 5.10 apresenta duas tabelas com os tempos, em milisegundos, da média de 1000 *pings*[6] entre cliente e servidor. A primeira apresenta as medidas realizadas em um computador Pentium II 300 Mhz com cliente e servidor executando na mesma máquina

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

(*standalone*). Esta medida permite avaliar o custo de processamento adicional relativo às computações do código SSL.

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

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

Figura 5.10- Comparação da latência do JaCoWeb com JacORB.

A segunda tabela apresenta as medidas realizadas, de forma distribuída, em uma rede local (com 10 Mbps *Ethernet*). Os tempos são medidos com um cliente em um computador Pentium II 300 Mhz e um servidor em um computador Pentium II 400 Mhz. Nesta situação, é considerado, além do custo de processamento do SSL, o custo relativo ao tempo gasto nas interações sobre a rede para o estabelecimento do contexto SSL. É interessante observar que a primeira chamada no *JaCoWeb* é relativamente alta (aproximadamente 1,7 segundos). Isto é o resultado da cifragem e decifragem com chaves públicas, que são requeridas para inicializar a primeira conexão do SSL. No entanto, para as chamadas (os *pings*) subsequentes, o tempo de aproximadamente 10 ms é considerado satisfatório.

A outra medida realizada foi o tempo de envio dos pacotes (Figura 5.11), com diferentes tamanhos, entre um cliente e um servidor nas mesmas configurações de *hardware*. O tamanho dos pacotes foram 1 Kbytes, 5 Kbytes, 10 Kbytes, 50 Kbytes e 100 Kbytes. Pode-se observar que os tempos medidos para o *JaCoWeb,* usando o SSL, são praticamente o dobro do obtido por um ORB convencional (JacORB). Estes resultados refletem o custo do suporte de segurança para criptografar mensagens transmitidas sobre a rede.

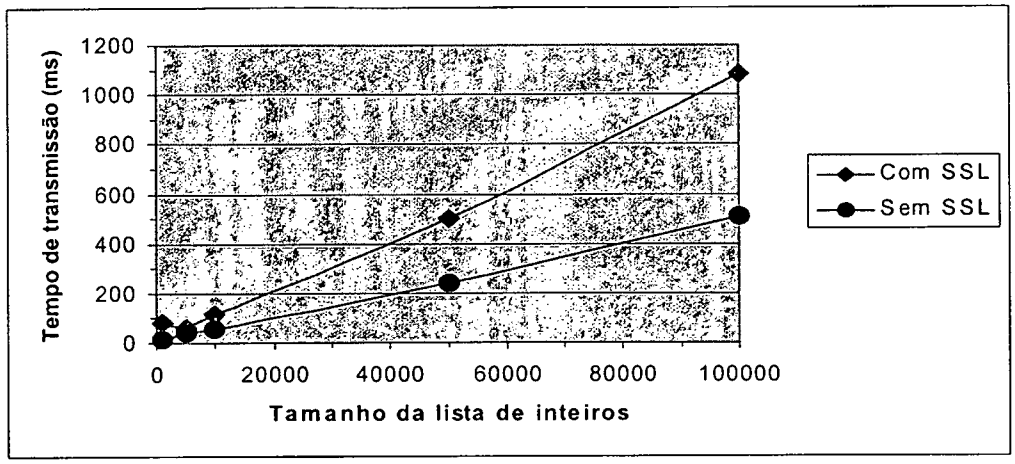


Figura 5.11- Tempo de envio de pacotes do *JaCoWeb* e do JacORB.

5.4.2- Avaliação do Esquema de Autorização Discricionária

A plataforma CORBA pode ser usada para integrar aplicações e sistemas, fornecendo a flexibilidade necessária para os ambientes de negócios que mudam dinamicamente, como os encontrados na Web. O uso do SSL, como tecnologia de segurança subjacente, garante às aplicações Web críticas às características desejáveis de integridade, confidencialidade e autenticidade das informações e recursos. Para estas aplicações ainda é preciso garantir que os recursos não serão usados por pessoas não autorizadas ou de formas não autorizadas. Neste caso, mecanismos de autorização, que também compõem o modelo CORBAsec, foram implementados visando garantir que somente usuários autorizados podem obter serviços de um sistema *internet banking* fictício.

Não é objetivo deste trabalho comparar o uso do modelo de segurança CORBA em aplicações Web, com os modelos de segurança que estão sendo atualmente usados nestas aplicações, como por exemplo os do Java. A intenção é comprovar através deste experimento que o padrão CORBA, combinado com o seu serviço de segurança, pode oferecer todas as características desejáveis de segurança para sistemas de larga escala, e que a implementação de um protótipo de um esquema de autorização discricionária é viável e confiável e que este pode exemplificar um possível trabalho cooperativo entre uma instituição bancária e pesquisas acadêmicas.

Diferentes testes foram realizados no sentido de verificar se a aplicação implementada fornecia os requisitos de segurança definidos (autenticidade, integridade e confidencialidade). Verificou-se que se um cliente do banco tenta desempenhar o papel de

gerente, por exemplo tentando apagar uma conta existente, ou se um usuário não autorizado tenta obter dados de uma determinada conta, as regras definidas para o controle de acesso impossibilita estas ações.

De forma a simular um ataque contra a integridade dos dados, foram alterados os dados quando estes estavam "navegando" na Internet e, como era de se esperar, a conexão SSL foi interrompida. Para analisar se a confidencialidade dos dados estava sendo preservada, monitorou-se a porta da conexão SSL. Como os dados estavam cifrados com o algoritmo simétrico em bloco (3DES_EDE), os dados analisados não se encontravam de forma legível e não permitiam a obtenção de informações sobre o tamanho dos blocos transferidos através de uma análise de tráfico, já que este algoritmo possui um mecanismo que obriga os blocos cifrados a terem o mesmo tamanho[7].

Com relação à implementação do esquema de autorização discricionária, avaliou-se o protótipo implementado de acordo com requisitos levantados na especificação CORBASec [36] e nas considerações dessa especificação apresentadas em [24]. Os requisitos, nos quais está baseada esta avaliação, são: transparência, simplicidade, reusabilidade, escalabilidade, flexibilidade e interoperabilidade.

A transparência oculta as funcionalidades (mecanismos) do sistema de segurança do usuário e do programador da aplicação, dando a impressão de que as invocações são atendidas localmente. O aspecto da simplicidade indica que o sistema de segurança deve ser fácil de ser entendido, usado e gerenciado. O *JaCoWeb Security* tenta atender a esses dois requisitos da melhor maneira possível, definindo um objeto *Security* (item 5.3.4), responsável pela inicialização e ativação de todos os mecanismos de segurança. A partir daí, as invocações são realizadas tal como num ORB convencional. No aspecto da facilidade de gerenciamento das configurações e políticas de segurança, o *JaCoWeb* é implementado de forma que as configurações de sistema sejam definidas estaticamente, em tempo de compilação, na classe *Enviroment*. No entanto, as políticas de segurança, para o controle de acesso às aplicações, podem ser definidas dinamicamente em tempo de execução através de uma interface de gerenciamento.

A reusabilidade é uma consideração importante, já que diminui o tempo de desenvolvimento e manutenção dos programas, tornando-os mais robustos e confiáveis. A

[7] Este mecanismo é conhecido como *padding* ou preenchimento

infra-estrutura de segurança do *JaCoWeb*, por ser baseada no modelo de objetos de serviço, pode ser facilmente adaptada para diferentes aplicações. Todos os serviços são descritos em IDL, padrão OMG, o que contribui não só no aspecto da reutilização de código como na portabilidade.

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

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

É claro que existe a dificuldade em atender completamente a cada um desses requisitos sem afetar o outro. O *JaCoWeb Security* se utiliza das opções que a especificação CORBASec oferece para atender, de forma equilibrada, esses requisitos. Novamente, o uso de interfaces IDL e o modelo de objeto de serviço (COSS) na implementação do *JaCoweb Security* nos permitiram atender melhor o aspecto da interoperabilidade. No entanto, como as especificações CORBASec padronizam uma grande parte de todas as abstrações de segurança (mecanismos, propriedades, políticas, estrutura de dados etc.), o aspecto da flexibilidade é bastante afetado. Contudo, isto não impede que extensões proprietárias sobre o modelo sejam implementadas na forma de objeto de serviço – sem alterar as características do ORB em si.

Em relação a comunicação, no estabelecimento do contexto de segurança através das tecnologias de segurança subjacentes, o *JaCoWeb Security* adota o SSL, juntamente com os protocolos de adaptação SSLIOP (*Secure Socket Layer Inter-ORB Protocol*) e SIOR (*Security-enhanced Inter-ORB Reference*), padrões da OMG. Isto significa que a interoperabilidade com outros ORBs só é alcançada quando ambos utilizam os mesmos

protocolos (SSLIOP) e tenham políticas de segurança "consistentes" [8]. Isto exclui ORBs que utilizam o protocolo de adaptação SECIOP (para SPKM, Kerberos e CSI-ECMA – no item 3) e DCEIOP (para DCE). No entanto, o *framework JaCoWeb Security* foi modelado de forma a permitir uma fácil adaptação às outras tecnologias de segurança, desde que tenha uma API disponível.

5.5- Conclusões do Capítulo

O protótipo implementado visou fornecer segurança a uma aplicação crítica muito utilizada pelos navegadores da Internet. Um sistema *internet banking* exige que somente usuários autorizados poderão ter acesso aos serviços do sistema e que as informações trocadas entre os clientes do banco e o servidor bancário necessitam ser criptografadas de forma que nenhum "bisbilhoteiro" tenha acesso a essas informações.

As tecnologias e ferramentas empregadas neste trabalho foram escolhidas devido à forte aceitação na Internet (SSL e o pacote iSaSiLk, linguagem Java e o JDK 1.2.1) e em ambientes CORBA (JacORB).

O protótipo do esquema de autorização discricionária garante à aplicação crítica selecionada boa parte das características desejáveis de segurança, tais como: autenticidade, integridade e confidencialidade. Apesar das dificuldades para seguir o modelo CORBAsec, devido às abstrações desta especificação, o protótipo implementado atingiu o seu objetivo que era comprovar a viabilidade e a confiabilidade do esquema de autorização proposto.

Para a integração de tecnologias de segurança ao ORB, foi proposto neste capítulo um *framework* geral que não compromete o aspecto da interoperabilidade do ORB como um todo. Este *framework* atua no sentido de adaptar uma tecnologia de segurança, tal como uma API que invoca os mecanismos de segurança do serviço de segurança subjacente.

Espera-se que a descrição da dinâmica dos modelos de autenticação, controle de acesso e associação segura, apresentados nesta dissertação, auxilie na implementação de aplicações práticas em ambientes de objetos distribuídos, que utilizam ORBs seguros, segundo o modelo CORBAsec, e desperte a comunidade científica nacional para o desenvolvimento dessa linha de pesquisa ainda aberta a modificações e melhorias.

Capítulo 6: Conclusão

Este trabalho apresenta e discute o serviço de segurança do padrão CORBA conforme especificado pela OMG, em 1997 (CORBAsec). A necessidade da integração de diversas tecnologias existentes nos ambientes distribuídos e heterogêneos pode ser satisfeita quando o *midleware* de comunicação CORBA é empregado. A plataforma CORBA apresenta os requisitos desejáveis às aplicações de sistemas abertos, tais como o aumento da reusabilidade, da portabilidade e da interoperabilidade. O CORBAsec veio adicionar o requisito segurança aos sistemas de larga escala, considerada hoje uma característica imprescindível em diversos ambientes de aplicações (sistemas bancários, comércio eletrônico, governo, redes privadas, companhias públicas, etc.).

Os modelos e a arquitetura de segurança propostos pelo *CORBAsec* foram projetados para atingir os principais objetivos de segurança das aplicações em ambientes heterogêneos. As funcionalidades de segurança desses modelos, como identificação, autenticação, autorização, auditoria de segurança, segurança da comunicação, não repudiação, administração das políticas e dos domínios de segurança possibilitam aos sistemas de objetos distribuídos, que empregam o CORBAsec, contornar ou diminuir as vulnerabilidades de seus sistemas [24].

O padrão CORBA e o seu serviço de segurança formam uma poderosa solução para integrar com segurança a infra-estrutura de tecnologia da informação ao mundo dos negócios tão vulnerável a modificações. Apesar disto, constata-se que ainda são poucas as pesquisas acadêmicas e produtos comerciais desenvolvidos que se utilizam desta combinação. De forma a apresentar e analisar esta solução em aplicações práticas um esquema de autorização discricionária foi descrito usando as abstrações propostas na especificação *CORBAsec*. Através da descrição da dinâmica dos modelos de autenticação, controle de acesso e associação segura, detalhados no capítulo 4, espera-se poder auxiliar a implementação de aplicações seguras em ambientes de objetos distribuídos, bem como contribuir para incentivar a pesquisa acadêmica nacional nesta área.

Com o objetivo de verificar a viabilidade e a confiabilidade do esquema de autorização discricionário, implementou-se um protótipo que visa fornecer segurança a um sistema *internet banking* fictício. Os resultados obtidos neste experimento comprovam a potencialidade do esquema proposto em fornecer segurança às aplicações críticas, em especial, às aplicações *Web*. Como apresentado no capítulo 5, o esquema de autorização discricionário proposto não foi totalmente implementado. É objetivo do projeto *JaCoWeb Security* aprimorar este protótipo inicial de forma que o controle de acesso seja fornecido em dois níveis, que o processo de autenticação seja completamente realizado e que o esquema empregue, além das políticas discricionárias, políticas obrigatórias. Uma avaliação qualitativa do esquema de autorização e do protótipo implementado com base no Critério Comum – CC faz parte ainda dos objetivos do projeto *JaCoWeb Security*.

Sugere-se ainda como continuação deste trabalho um estudo do processo de auditoria e do processo de não-repudiação que, com certeza, acresceria ainda mais segurança ao esquema de autorização proposto no projeto *JaCoWeb Security*. Os problemas de interoperabilidade, de gerência e administração de domínios precisam ser discutidos de forma a acompanhar as RFP's (*Request for Proposal*) sugeridas pela OMG.

No contexto das aplicações críticas desenvolvidas para a Internet, há, além do serviço de segurança CORBA, os modelos de segurança das plataformas Web e Java cuja combinação ainda necessita de pesquisas aprofundadas, já que estas plataformas estão se constituindo em padrões *de facto* para a programação distribuída na Internet.

Acredita-se que, assim como este trabalho ilustra a possibilidade de uma ligação entre empresa e universidade, outras soluções que visam fornecer segurança a sistemas distribuídos podem ser pesquisadas dentro das universidades. Diversas pesquisas acadêmicas estão sendo realizadas nesta área. O ambiente de negócios vem exigindo a segurança da informação. Então por que não estabelecer uma maior integração entre empresa e universidade ?

REFERÊNCIAS BIBLIOGRÁFICAS

[1] 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/).

[2] AMOROSO, Edward, *Fundamentals of Computer Security Technology*. Englewood Cliffs, NJ: Prentice-Hall, 1994.

[3] BAKER, S., *CORBA Distributed Object: using Orbix*. Addison-Wesley, 1997.

[4] BELL, D. E., LAPADULA, L. J., *Secure computer systems: Mathematical foundations and model*. Tech. Rep. M74-244, MITRE Corp, October 1974.

[5] BIBA, K. J. *Integrity Considerations for Secure Computer Systems*. Technical Report ESD-TR 76-372, MITRE CO., april 1997.

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

[7] CHIZMADIA, D., CORBAsec: Securing Distributed Systems – Find out what CORBAsec has to offer for security-aware applications. *Software Development Magazine*, May 1999.

[8] CLARK, D. D. and WILSON, D. R., A comparision of Commercial and Military Computer Security Policies. *In Proc. of the 1987 Symposion on Research in Security and Privacy*, IEEE Society Press, pages 184, 194, Okland, Californie, mai, 1997.

[9] COMMON CRITERIA PROJECT SPONSORING ORGANIZATIONS. *Common Criteria for Information Technology Security Evaluation*, In Part2: Security Funcional Requirements, Doc. CCIMB-99-032, Version 2.1, August 1999. (http://csrc.nist.gov/cc/ccv20/ccv2list.htm#CCV21)

[10] COULOURIS, G., DOLLIMORE, J., *Distributed Systems - Concepts and Design.* Addison-Wesley, 1994.

[11] FORD, W. *Computer Communications Security - Principles, Standard Protocols and Techniques*. Prentice-Hall, 1994.

[12] FREIER, A. O., KARLTON, P. and KOCHER, P.C., *Secure Socket Layer 3.0*, Internet Draft, Nov. 1996

[13] GARFINKEL, S., SPAFFORD, G., *Practical Unix and Internet Security.* O'Reilly & Associates Inc, 1996.

[14] GARFINKEL, S., SPAFFORD, G., *Web Security & Commerce*. O'Reilly & Associates Inc, 1997.

[15] HARRISON, M. A., RUZZO, W. L., ULMAN, J.D., Protection in operating systems. *Communications of the ACM*, Vol. 19, No. 8, October 1976, pp. 461-471.

[16] HARTMAN, B., *CORBA Security – Beta Product Releases from ICL and Iona*, Distributed Computing Monitor Feature Report Summary, October, 1997. (http://www.psgroup.com/features/1997/fl097d.htm)

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

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

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

[20] KARJOTH, G. Authorization in CORBA Security, In *Proceedings of the Fifth ESORICS*, Lecture Notes in Computer Science, pp. 143-158, Springer-Verlag, Berlin Germany, September 1998.

[21] LAMPSON, B. W. Protection, *Proc. 5th Princeton Symp. Information Sciences and Systems* (March 1971), pp. 437-443

[22] LANDWEHR, C. E., Formal models for computer security. *Computing Surveys ACM*, Vol. 13, No. 3, September 1981, pp. 247-278.

[23] LANG, U., *Security Aspects of the Common Object Request Broker Architecture*, 1997. MSc Information Security – University of London.

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

[25] LISTA de discussão sobre o *corba-security* [online]. Disponível: Email: corba-security@cs.fiu.edu [mensagem capturada em 19 aug. 1999]

[26] LISTA de discussão sobre o *corba-security* [online]. Disponível: Email: corba-security@cs.fiu.edu [mensagem capturada em 8 fev. 2000]

[27] MÓDULO SECURITY SOLUTIONS S.A. *5ª Pesquisa Nacional sobre Segurança da Informação*. 1999. http://www.modulo.com.br/noticia/pesquisa.htm

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

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

[30] NORMALISATION – Java en route vers le standard ISO, *Le Monde Informatique*, n. 743, 21 novembre 1997.

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

[32] Object Space, *VoyagerSecurity 3.2 – Developer Guide*, 1999. http://www.objectspace.com/products/documentation/voyager_html_docs/security/index.htm

[33] OMG Document. *OMG White Paper on Security*. OMG Security Working Group, April 1994.

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

[35] OMG, *CORBAservices: Common Object Services Specification*, OMG Document March, 1997.

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

[37] OMG, *Common Secure Interoperability Version2*, Doc. orbos/99-01-10, January, 1999.

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

[39] PeerLogic, *DAIS Security – A PeerLogic White Paper*, 1998.

[40] PeerLogic, *LiveContent BROKER 3.2 — An Introduction*, 1999. http://www.peerlogic.com/products/products.html

[41] Promia, *SecureBroker*. http://www.promia.com/products/securebroker.html

[42] SÊMOLA, M. Já é hora de pensar em e-security. *IDG Now*, setembro 1999. http://www.modulo.com.br/noticia/a-esec.htm

[43] SÊMOLA, M. Internet banking: nem tudo são flores. Disponível em: http://www.modulo.com.br/noticia/a-intban.htm

[44] SNYDER, L., Theft and conspiracy in the take-grant protection model. *Journal of Computer and System Sciences*, Vol. 23, No. 3, December 1981, pp. 333-347

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

[46] Syntegra on behalf of the Common Criteria Project Sponsoring Organisations. *Common Criteria – Na Introduction*, 1998 (http://csrc.nist.gov/cc/info/cc_brochure.pdf).

[47] VINOSKY, S., CORBA: Integrating Diverse Applications Within Distributed Heterogeneous Environments. *IEEE Communications Magazine*, Vol. 14, No.2, February 1997.

[48] VOYDOCK, V.L., KENT S.T. Security Mechanism in High-Level Network Protocols. *Computing Surveys*, Vol. 15, No. 2. June 1983.

[49] WESTPHALL, C.M. *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, Brasil, 1998

[50] WESTPHALL, C.M., and FRAGA, J.S. 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.

[51] WESTPHALL, C.M., FRAGA, J.S, WANGHAM, M.S., *PoliCap* - Um Serviço de Política para o Modelo CORBA de Segurança. Submetido ao SBRC 2000, 2000.