UNIVERSIDADE FEDERAL DE SANTA CATARINA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

N'guessan Désiré

UM MODELO DE GERÊNCIA DE SEGURANÇA BASEADO EM OBJETOS DISTRIBUÍDOS

Dissertação submetida à Universidade Federal de Santa Catarina como parte dos requisitos para a obtenção do grau de Mestre em Ciência da Computação

Prof. Dr. Carlos Becker Westphall
Orientador

Florianópolis, Abril de 2000

UM MODELO DE GERÊNCIA DE SEGURANÇA BASEADO EM OBJETOS DISTRIBUÍDOS

N'guessan Désiré

Esta Dissertação foi julgada adequada para a obtenção do título de Mestre em Ciência da Computação Área de Concentração SISTEMAS DE COMPUTAÇÃO e aprovada em sua forma final pelo Programa de Pós-Graduação em Ciência da Computação

Prof. Dr. Carlos Becker Westphall
Orientador

Prof. Dr. Fernando Álvaro Ostuni Gauthier
Coordenador do Curso de Pós-Graduação
em Ciência da Computação

Banca Examinadora:

Prof. Dr. Carlos Becker Westphall
Presidente

Prof. Dr. João Bosco Mangueira Sobral
Membro

Prof. Dr. Bernardo Gonçalves Riso
Membro

Dra. Mirela Sechi Moretti Annoni Notare
Membro

Dedicatória

À Esaïe W. Brice.

À Lucas de Sousa N'guessan.

" Je vous aime ... "

Agradecimentos

Agradecimento aos meus pais N'guessan Yoffoua e Gnagna Madeleine, por terem me dado uma educação exemplar e por apresentarem um exemplo de caráter e honra que sempre procurei seguir. Sem eles não teria alçando este mérito. Sinto muito saudade.

Às minhas irmãs: Cathérine, Rosalie e Georgette pelo constante carinho e apoio que me deram.

Ao prof. Dr. Carlos Becker Westphall, pois além de me orientar e apoiar neste trabalho, foi também um grande e ilustre amigo. Nos momentos de desespero, ele sempre soube dar o toque mágico. Me lembro sempre desta frase de estímulo que ele passava à seus orientandos " *Tous les jours on va de mieux en mieux à tous les points de vue*". Devo muito a ele.

Aos membros da banca examinadora, pelas criticas, sugestões, incentivos e elogios, em especial à Dra. Mirela Sechi Moretti Ennoni Notare, pela amizade, dedicação e empenho sem os quais este não teria se realizado.

Aos membros do LRG (Laboratório de Rede e Gerência) pelo carinho e acompanhamento deste trabalho.

À Dra. Sandra Moraes Dias, que nos momentos mais difíceis sempre esteve firme ao meu lado demostrando o verdadeiro sentido de uma grande amizade.

Aos colegas incansáveis do curso da pós-graduação, à Gilmari Ana Conte, à todos que acreditaram na minha capacidade e contribuíram de maneira direta ou indireta no sucesso deste trabalho.

Finalmente agradeço à mim e à Deus, pois sem ele, nada disso seria possível.

Sumário

Lista de Figuras

Lista de Quadros

Glossário de Abreviaturas

API : Application Programme Interface

CORBA : Common Object Request Broker Architecture

DII : Dynamic Invocation Interface

DES : Data Encryption Standard

DSA : Digital Signature Algorithm

DSI : Dynamic Skeleton Interface

ESIOP : Environment Specific-Inter-ORB Protocol

GIOP : Generic Inter-ORB protocol

GRG : Grupo de Rede e Gerência

GUI : Graphic User Interface

ID : Identity

IDEA : International Data Encryption Algorithm

IDL : Interface Definition Language

IIOP : Internet Inter-ORB Protocol

ISO : International Standars Organization

ITU -T : International Telecommunicatons Union - Telecommunications Standardization Sector

LRG : Laboratório de Redes e Gerência

OMA : Object Management Architecture

OMG : Object Management Group

ORB : Object Request Broker

PC : Personal Computer

POA : Portable Object Adapter

RMI : Remote Method Invocation

RSA : Rivest-Shamir-Adheman

SEMICO : Seminário de Computação

SETWeb : Sistema de Extrato Telefônico via Web

SSH : Security Shell

SSL : Security Socket Layer

SSTCC : Sistema de Segurança para Telecomunicação contra Clonagem de Celulares

TCP/IP : Tranfer control protocol/ Internet protocol

VPN : Virtual Private Network

UFSC : Universidade Federal de Santa Catarina

WWW : World Wide Web

Resumo

Este trabalho explora a gerência de segurança no âmbito de gerenciamento de sistemas distribuídos. Nesse contexto um mecanismo de autenticação e de autorização foi implementado. A implementação considera as tecnologias da *Internet (Java, WWW and Applet Java)* e sistemas distribuídos baseados em objetos distribuídos segundo o OMG-CORBA (Object Management Group - Common Object Request Broker Architecture).

A abordagem baseia-se em pseudo-objeto e nos interceptores do Visibroker da Inprise (um consórcio fabricante de produto CORBA). A idéia é que, após fornecer seu ID de usuário e sua senha para entrar em uma sessão de aplicação, o cliente obtenha um identificador de sessão a partir de uma base de segurança, ou *ticket*. Este *ticket* contém o ID (IDentity) do usuário em uma forma criptografada. O cliente envia o *ticket* como sendo o nome do *principal* [01]. Este pseudo-objeto junto com a requisição do cliente é enviado pelo ORB (Object Request Broker, ou Analisador de Requisições a Objetos) toda vez que o cliente faz uma invocação em um objeto. Quando a requisição chega ao servidor ela é interceptada por uma Autoridade de Segurança e o *ticket* do "*principal*" é enviado à base de segurança para um controle de acesso.

O mecanismo é fácil de operar, independente do serviço de segurança apresentado pelo CORBA, proporciona segurança no acesso aos objetos-servidores distribuídos e aos seus usuários e tem total controle sobre o acesso aos objetos-servidores, o que implica na habilidade para a realização de auditoria no acesso aos objetos-servidores e para saber quem invocou quais operações e de quais objetos.

O mecanismo implementado foi validado pela sua aplicação no protótipo SETWeb (Sistema de Extrato Telefônico via Web) que faz parte do SSTCC (Sistema de Segurança para Telecomunicação Contra Clonagem de Celulares), um sistema composto de vários projetos que estão sendo desenvolvidos no Laboratório de Redes e Gerência (LRG) da Universidade Federal de Santa Catarina (UFSC).

Palavras-Chaves: Sistemas Distribuídos, Gerência de Segurança, CORBA, WWW, Java.

Abstract

This work explores security management, in the ambit of management of distributed systems. In this context an authenticaton and authorization mechanism was implemented. The implementation considers Internet technologies like Java, WWW and Applet Java and distributed systems based on objects distributed according to OMG-CORBA (Object Management Group - Common Object Request Broker Architecture).

The approach bases on pseudo-object and the interceptors of Visibroker of Inprise (a manufacturing consortium of product CORBA). The idea is that the client after supplying its user ID and its password to enter in an application session obtains from a base of security a session badge, or ticket. This ticket contains ID (IDentity) of the user in some encrypted form. The client sends the ticket as being the name of the principal[01]. This pseudo-object with the requisition of the client is sent by the ORB (Object Request Broker, or Analisador of Requisitions to Objects) all time that the client makes an invocation in an object. When the requisition arrives to the server it is intercepted by a Security Authority and the ticket of the principal is sent to a base of security for an access control.

The mechanism is easy to implement, independent of the security service of CORBA Services. It provides a larger security in the access to the objects distributed servers and its users, and has a total control on the access to the objects servers as demonstrated by the obtained results. What implies in the ability for auditing the access to the objects servers and to know who invoked which operations and of which objects.

The implemented mechanism was validated by its application in SETWeb prototype (System of Phone Extract saw Web) that does part of SSTCC (Security System for Telecommunication Against Cellular Cloning), a system composed of several projects that are being developed in the Network and Management Laboratory (LRG) of Santa Catarina's Federal University (UFSC).

***Key words**: Distributed systems, Security management, CORBA, WWW, Java.*

Capítulo 1

Introdução

1.1 Motivação

No principio da criação e da difusão das redes de computadores, havia a necessidade de se desenvolver uma tecnologia de baixo custo, que atendesse as necessidades prementes de comunicação, e que fosse simples o suficiente para que se difundisse rapidamente. Isto de fato ocorreu, mas a simplicidade e o baixo custo excluíram os investimentos em tecnologia de segurança [29].

Nos últimos anos, em função das novas tecnologias que vêm surgindo e da necessidade de se obter uma quantidade de informações imprescindíveis à realização de diversos serviços, tornou-se necessária a criação de ambientes de processamento de informações distribuídos. Esses ambientes oferecem e viabilizam o acesso local ou remoto às informações de forma transparente e homogênea para o usuário. Com isso, surgiram problemas de distribuição e comunicação, dificultando a interoperabilidade e a interconectividade na integração de sistemas distribuídos. A forma mais indicada para estabelecer interconectividade entre recursos é o modelo Cliente/Servidor, que organiza os sistemas distribuídos como um número de servidores distribuídos e oferece um conjunto de serviços para os clientes

através da rede. Essas novas abordagens introduziram também novos problemas de segurança, que implicaram em novos conceitos e modelos de segurança [25].

A interconexão dos sistemas e das redes cresceram, tornando os sistemas potencialmente acessíveis por um número de usuários cada vez maior. O uso dessas redes para transferência de informações sensíveis, privadas ou confidenciais cresceu, e desta forma, a segurança tornou-se um requisito básico nesses sistemas. Mas infelizmente a segurança foi sempre descartada quando comparada com a performance [11], porém com a interconexão cada vez maior do mundo através da Internet as organizações estão começando a se preocupar com a segurança em redes. A segurança em sistemas distribuídos é muito importante [11].

Os trabalhos encontrados sobre a segurança em rede e sistemas distribuídos têm se caracterizado como embrionários. Os modelos e técnicas de segurança evoluíram muito pouco [10].

Os protocolos de segurança implementados pelos produtos comerciais ainda são incompletos, por exemplo, as implementações do *Security Socket Layer* (SSL) fornecem uma criptografia e uma autenticação somente do servidor mas não do usuário [25]. O mecanismo baseado em chaves privadas do Kerberos ainda não é viável quando se trata de um número maior de chaves de usuários [25]. O Kerberos é muito usado porém ele apresenta algumas limitações de segurança [35, 36, 37], o problema mais grave é a vulnerabilidade do seu servidor. Um intruso, a partir do credencial do administrador (root), pode obter chaves secretas críticas. Com estas chaves em mão, ele pode revelar toda a informação protegida do servidor Kerberos. Além disso o Kerberos e outros *Frameworks* de segurança, tais como, POXIX e ISO7498-2 não suportam ambientes distribuídos orientados a objetos [38, 39].

Muitos destes protocolos de segurança apresentam uma taxa de *overhead* (atraso) muito grande devido à complexidade dos algoritmos de criptografia (cifradores) empregados

nos seus mecanismos e pelo fato de serem implementados na camada de rede como é o caso do SSL [31, 32].

1.2 Objetivos

Com base na motivação acima relatada, o objetivo geral deste trabalho é propor e implementar, no âmbito da gerência de segurança em sistemas distribuídos baseados em objetos, um mecanismo de autenticação e autorização que apresente as seguintes características:

- Fácil de implementar e operar, ser implementado na camada de aplicação para resolver o problema de overhead apresentado pelos protocolos de segurança existentes;

- Proporcionar segurança no acesso de objetos-servidores distribuídos e aos seus usuários;

- Ter total controle sobre o acesso aos objetos-servidores, o que implica na habilidade para auditar o acesso aos objetos-servidores a fim de saber quem invocou quais operações e de quais objetos.

Para alcançar este objetivo geral foram definidos os seguintes objetivos específicos:

- Estudo da tecnologia orientada a objetos, das tecnologias da *Internet* e da arquitetura CORBA, visando demonstrar que as três tecnologias formam um relacionamento perfeito para construção de aplicação distribuída heterogênea;

- Estudo do modelo de referência de segurança definido pelo OMG ;

- Implementação do mecanismo de Autenticação e Autorização em uma aplicação distribuída, usando *Java, Web* e CORBA, visando demonstrar e validar o presente trabalho.

1.3 Organização do Trabalho

Este trabalho está organizado da seguinte forma:

- No primeiro capítulo, apresenta-se a introdução onde são explicados a motivação e os objetivos deste trabalho;

- No segundo capítulo, abordam-se as idéias básicas da Tecnologia Orientada a Objetos. Nesse capítulo são definidos os conceitos de objeto, classe, métodos, solicitação (*request*), encapsulamento e herança. Essas idéias constituem um embasamento teórico para melhor entender a implementação do mecanismo proposto, totalmente baseada no paradigma da programação orientada a objetos;

- O terceiro capítulo apresenta a computação distribuída na era da Internet. Salienta-se a importância da tecnologia orientada a objetos na conceituação e desenvolvimento de aplicações distribuídas. Descreve-se a arquitetura CORBA, uma arquitetura para computação distribuída baseada em objetos. Esta arquitetura oferece uma série de benefícios que permitem a criação e a integração de aplicações em ambientes distribuídos heterogêneos;

- No quarto capítulo, focalizam-se os aspectos das tecnologias da Internet como *Java, Applet Java Web Browsers* e de segurança dos *Firewalls* que, juntos com a arquitetura CORBA, permitem desenvolver aplicações heterogêneas que suportam interoperabilidade;

- O capítulo 5 apresenta a segurança em sistema de informações, onde discutem-se os diferentes tipos de ataques que criam uma necessidade de mecanismos e serviços de Segurança de Rede. Apresentam-se sucintamente esses serviços, que consistem em medidas de segurança que permitem deter, prever, detectar e corrigir violações que envolvem a transmissão de informações;

- No capítulo 6, salienta-se o modelo de referência de segurança do CORBA segundo o OMG;

- No sétimo capítulo, descreve-se a implementação do mecanismo de autenticação e autorização proposto; e

- Finalmente, o último capítulo apresenta as conclusões e as perspectivas futuras do trabalho.

1.4 Publicações

Este trabalho produziu artigos que foram publicados e selecionados para os seguintes eventos:

- VIII SEMICO Seminário de Computação, ocorrido de 09-11 de outubro de 1999 em Blumenau - SC – Brasil. E publicado nos anais do evento nas paginas 23-30;

- IDEAS'00 III Jornadas IberoAmericanas de Ingeniería de Requisitos y Ambientes de Softwares, evento que ocorrerá de 5 – 7 de abril 2000, em Cancúm, México.

Capítulo 2

Conceitos da Tecnologia Orientada a Objetos

2.1 Introdução

As idéias fundamentais da tecnologia orientada a objetos são extremamente importantes para definir as características de uma aplicação distribuída baseada em objeto. Essas idéias incluem:

- Objetos e Classe (Sendo uma classe a implementação de um objeto);
- Métodos;
- Solicitações (Request);
- Encapsulamento;
- Herança.

2.2 Objetos

O conceito fundamental de objeto significa uma abstração de alguma coisa no domínio de um problema ou em sua implementação, refletindo a capacidade de um sistema manter informações sobre ela, interagir com ela, ou ambos [24]. Em outras palavras um objeto é qualquer coisa real, ou abstrata, a respeito da qual armazenamos dados e os métodos que os manipulam.

2.3 Métodos

Os métodos especificam a maneira pela qual os dados de um objetos são manipulados. Os métodos de um tipo de dados referenciam somente as estruturas de dados desse tipo de objeto [25].

Para usar a estrutura de dados de outros objetos, eles devem enviar uma mensagem a esse objeto. Por exemplo, um método associado a um tipo de objeto *Fatura* poderia computar o *total de uma fatura* ou *transmitir a fatura a um cliente*, ou ainda *verificar periodicamente se a fatura foi paga e adicionar juros a ela, caso não tenha sido pago.*

2.4 Encapsulamento

O ato de empacotar ao mesmo tempo dados e métodos é denominado encapsulamento. Um objeto esconde seus dados de outros objetos e permite que seus dados sejam acessados por intermédio de seus próprios métodos. Isso significa ocultação de informações (*information hiding*). O encapsulamento protege os dados de um objeto contra as adulterações. Se todos os programas pudessem acessar os dados de quaisquer maneiras que os usuários desejassem acessar, os dados poderiam ser facilmente adulterados ou usados de forma inadequada. O encapsulamento protege os dados do objeto do uso arbitrário e não-intencional.

O encapsulamento oculta os detalhes de sua implementação interna aos usuários de um objeto. Os usuários entendem quais operações do objeto podem ser solicitadas, mas não conhecem os detalhes de como as operações estão executadas. Todos os aspectos específicos dos dados do objeto e a codificação de suas operações são mentidas numa caixa preta fora de vista. O encapsulamento é importante porque separa a maneira como um objeto se comporta da maneira como ele é implementado. Isso permite que as implementações dos objetos sejam modificadas sem exigir que os aplicativos que as usam sejam também modificados.

É mais fácil de modificar um programa usando-se o encapsulamento porque um tipo de objeto é modificado de uma só vez. Se um tipo de objeto for modificado, somente os métodos e as estruturas de dados associados a esse tipo de objeto são afetados, e usualmente apenas alguns desses métodos e estrutura de dados o são. O comportamento do tipo de dados de objetos pode ser modificado e testado, independentemente de outros tipos de dados de objetos. A Figura 2.1 ilustra a estrutura de um objeto. A estrutura de dados no centro pode ser usada somente com os métodos do anel externo.

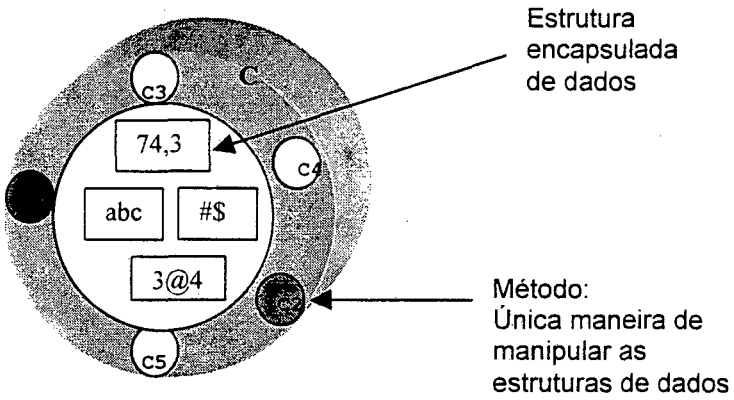

Figura 2.1 - Estrutura de um Objeto.

Na Figura 2.1 o objeto C encapsula a estrutura de dados e os métodos: c1, c2, c3, c4 e c5. O objeto é manipulado por métodos que implementam as operações permitidas. A estrutura de dados pode ser usada somente com os métodos. Essa restrição protege os dados contra adulteração.

2.5 Mensagens

Para fazermos um objeto realizar alguma coisa, enviamos uma solicitação. Esta faz com que uma operação seja invocada. Essa operação executa o método apropriado e, opcionalmente retorna uma resposta. A mensagem que constitui a solicitação contém o nome do objeto, o nome da operação, e às vezes, um grupo de parâmetros. A programação

orientada a objetos é uma forma de projeto modular, em que o mundo freqüentemente é imaginado em termo de objetos, operações, métodos e mensagens passadas entre esses objetos, conforme ilustrado na Figura 2.2 [25]. Uma mensagem é uma solicitação para executar a operação indicada sobre determinados objetos e retornar os resultados. Conseqüentemente as implementações orientadas a objetos referem-se a mensagens como solicitações.

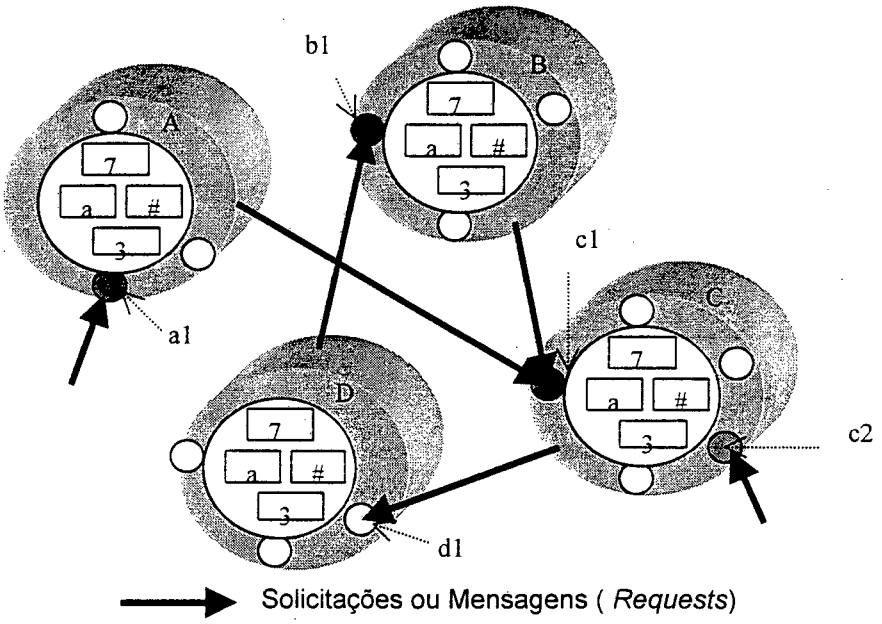

Figura 2.2 - Solicitações de Métodos entre Objetos.

Na Figura 2.2 os objetos comunicam-se através de solicitações. Uma solicitação é uma mensagem especificando que uma operação indicada seja executada, usando um ou mais objetos e, provavelmente, retornando um resultado. Nessa figura, por exemplo, os objetos A e B estão solicitando através de mensagens a execução da operação c1 do objeto C.

Os objetos podem ser muito complexos, pois um objeto pode conter outros objetos (subobjetos); estes, por sua vez, pode conter outros objetos (subsubobjetos) e assim por diante. Um projetista que usa o objeto não precisa conhecer sua complexidade interna; precisa somente saber como se comunicar com ele e como ele responde.

2.6 Classe

O termo classe refere-se à implementação em software de um tipo de objeto. Tipo de objeto é uma noção conceitual, ele especifica uma família de objetos sem estipular como o tipo e os objetos são implementados [25]. Os tipos de objetos são especificados durante a análise orientada a objeto. Porém quando se implementam tipos de objetos outros termos são usados: na linguagem de programação *Modulo*, os tipos de objetos são implementados como módulos, enquanto a linguagem *Ada* usa a palavra pacote. Nas linguagens orientadas a objetos, os tipos são implementados com classe.

A implementação da classe especifica a estrutura de dados para cada um de seus objetos (veja a Figura 2.1). Por exemplo, uma classe *Empregado* incluiria dados sobre isenções, cargo, salário, telefone dentre outros. Além disso, cada classe define um conjunto de operações permissíveis que permitem acesso e modificação de dados dos objetos. Uma classe de *empregado* poderia incluir operações, tais como *contratar*, *promover*, *mudar o telefone*. Os detalhes sobre as operações são especificados pela classe.

2.7 Herança

Um tipo de objeto de alto nível pode ser especificado com a utilização de um tipo de objeto de nível mais baixo. Um tipo de objeto pode ter subtipos, por exemplo, o tipo de objeto Pessoa pode ter os subobjetos Pessoa Civil e Pessoa Militar. Pessoa Militar pode ter subobjeto Oficial Alistado assim por diante. Há uma hierarquia de tipos de objetos, subobjetos, subsubobjetos etc... Uma classe implementa o tipo de objeto. Uma subclasse **herda** as propriedades de sua classe-mãe (ou superclasse), uma subsubclasse herda as propriedades da subclasse e assim por diante. Uma subclasse pode herdar as estruturas de dados e todos os métodos ou alguns dos métodos de sua superclasse. Ela também tem métodos e, às vezes, tem tipos de dados próprios. A Figura 2.3 mostra uma classe A e uma subclasse B, onde a subclasse tem os mesmos métodos de sua superclasse, mas também o método *b1* que é seu próprio método. Às vezes uma classe herda propriedades de mais uma classe. Isso é dominado de herança múltipla.

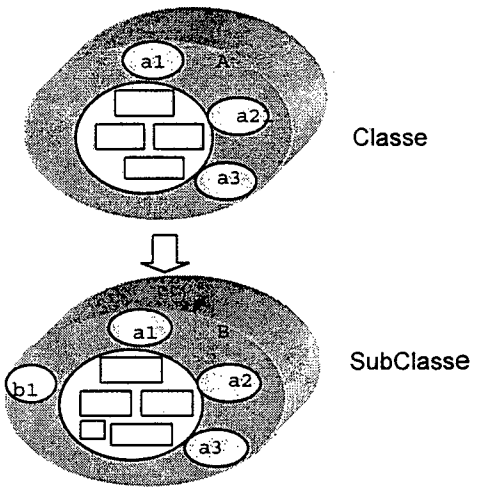

Figura 2.3 – Herança.

Uma classe pode ter suas próprias estruturas de dados e métodos bem como herdá-la de sua superclasse. Por exemplo, a subclasse B herdou os métodos: *a1, a2* e *a3* da classe A.

Capítulo 3

Computação de Objetos Distribuídos

3.1 Introdução

Nos últimos 15 anos ocorreram muitas mudanças no desenvolvimento e na manutenção dos sistemas de informações das empresas. Este período começou com os sistemas *mainframes* monolíticos, onde cada sistema tinha seu próprio gerenciamento de apresentação, sua própria lógica de aplicação e de acesso a dados. Cada interface proprietária de acesso a dados era depende de um único sistema de dados. Não era possível um sistema compartilhar dados com outros sistemas devido aos problemas de sincronização e integridade de dados, portanto cada sistema tinha que armazenar cópias privadas de seus próprios dados para *backup*. Esses sistemas monolíticos eram muito ineficientes e apresentavam um alto custo. Eles foram, então, substituídos pelos modelos cliente/servidor. Esta mudança foi possível pela convergência das tecnologias de redes, do baixo custo dos PCs, das GUIs (*Graphic User Interface*) e dos bancos de dados relacionais. A computação cliente/servidor simplificou o desenvolvimento e a manutenção das aplicações complexas, dividindo os sistemas monolíticos centralizados em componentes clientes e servidores [31].

No modelo cliente/servidor, o cliente solicita um determinado serviço. Um ou vários processos chamados de servidores são responsáveis pelo fornecimento de serviços ou por responder às solicitações do cliente. Os serviços são acessados via interface. Ao receber uma solicitação válida, o servidor executa a operação apropriada e entrega a resposta de

volta ao cliente. Este tipo de interação é conhecido como *Request/Reply*. O modelo cliente/servidor aumentou a disponibilidade e otimizou o custo, mas não resolveu o problema da reusabilidade de códigos. Para reutilizar os códigos de um módulo, os segmentos de códigos similares eram simplesmente copiados e adaptados a outro módulo. A manutenção dos módulos similares era feita separadamente em cada módulo. Muitas soluções cliente/servidor simplesmente dividiram o sistema monolítico *mainframe* em dois.

As soluções apresentadas pelo modelo cliente/servidor são, ainda, incompletas. Os problemas associados a, por exemplo reusabilidade, escalabilidade (expansão), a interoperabilidade, autonomia e mobilidade continuam existindo, tornando o desenvolvimento e a manutenção das aplicações cliente/servidor com missão crítica muito difíceis. As tecnologias de objetos distribuídos e da Internet convergem no mesmo caminho, no sentido de proporcionar uma melhor solução aos problemas acima relatados [31].

3.2 Objetos Distribuídos

A tecnologia de objetos distribuídos transformou as aplicações cliente/servidor monolíticas em componentes autogerenciáveis, ou objetos, que podem interoperar em redes e sistemas operacionais diferentes. Os objetos são implementados segundo os conceitos da programação orientada a objetos, sendo executados em ambientes que suportam serviços como localização transparente de objetos, invocação de métodos em objetos locais ou remotos e migração de objetos, usando um ORB (*Object Request Broker*) [02].

Essa tecnologia permite às organizações de Tecnologia de Informações construir infra-estruturas que se adaptam às mudanças e às oportunidades do mercado. Desde 1997, a tecnologia de objetos vem se tornando o paradigma mais usado para conceituar e desenvolver as aplicações [30]. A tecnologia de objetos distribuídos apresenta várias vantagens como, por exemplo, a reusabilidade e a escalabilidade. Associada ao paradigma cliente/servidor ela permite criar aplicações distribuídas que proporcionam uma melhor

forma para a integração no processamento das informações distribuídas, ou aplicações que suportam interoperabilidade. As aplicações distribuídas, entretanto trouxeram novas requisições: operar em ambientes de computação heterogênea, funcionar em uma variedades de *hardwares* e plataformas diferentes, integrar as tecnologias antigas com as novas. Elas também requerem um alto grau de disponibilidade e de performance, uma facilidade de administração, e sobretudo uma integridade dos dados ou seja a segurança de dados [30].

Para atender a essas novas requisições, existem atualmente vários ambientes e ferramentas para desenvolvimento de aplicações distribuídas orientadas a objetos. Entre essas tecnologias que suportam o desenvolvimento de aplicações baseadas em objetos, as mais conhecidas no mercado são: CORBA que é a proposta do OMG, o DCOM a solução da *Microsoft* e o Java-RMI do *JavaSoft*. Um ponto comum entre as diferentes abordagens para sistemas distribuídos orientados a objetos, apresentadas por essas tecnologias é a existência de um elemento responsável por disponibilizar, de modo transparente, os objetos das aplicações servidoras às aplicações clientes [04]. No estado da arte o CORBA se destaca melhor, portanto, neste trabalho, apresenta-se a proposta do OMG. A próxima seção (seção 3.3) descreve a arquitetura CORBA, uma arquitetura para computação distribuída baseada em objetos.

3.3 CORBA

O OMG é um consórcio internacional, criado em 1989, que agrupa atualmente mais de 850 membros do mundo da informática: os construtores tais como (IBM, Sun), os produtores de programas tais como (Netscape, Inprise ou Ex-Borland/Visigenic, IONA Tech.), os usuários (Boeing, Alcatel) por exemplo e as instituições e universidades (Nasa, Inria, Lift) por exemplo. O objetivo desse grupo é definir padrões para a integração de aplicações distribuídas heterogêneas, a partir de tecnologias orientadas a objeto. Portanto, os conceitos chaves enfatizados são: reusabilidade, interoperabilidade, e portabilidade dos componentes das aplicações distribuídas. As especificações do OMG são reconhecidos pelo

ISO (*International Standars Organization*) e pelo ITU-T (*International Telecommunicatons Union - Telecommunications Standardization Sector*). O elemento principal da visão do OMG é o CORBA [04]. No final de 1994, o OMG lançou o CORBA versão 2.0 que inclui o protocolo IIOP. No presente trabalho foi usada a versão 2.3. O que faz do CORBA uma ferramenta importante é o potencial de assumir o lugar de um *middleware*[1] cliente/servidor, usando objetos como forma de unificação de aplicações existentes. A utilização do CORBA faz com que o sistema inteiro seja auto-descritivo, pois a especificação dos serviços é feita de forma separada da implementação, o que permite a incorporação de sistemas existentes, independência de plataforma e de linguagem de programação. Um objeto distribuído CORBA pode viver em qualquer lugar da rede, sendo acessado por clientes remotos através de métodos de invocação. Veja a Figura 3.1 onde um cliente implementado na forma de *applet* pode invocar um objeto no servidor remoto de uma rede de empresa via a Internet.

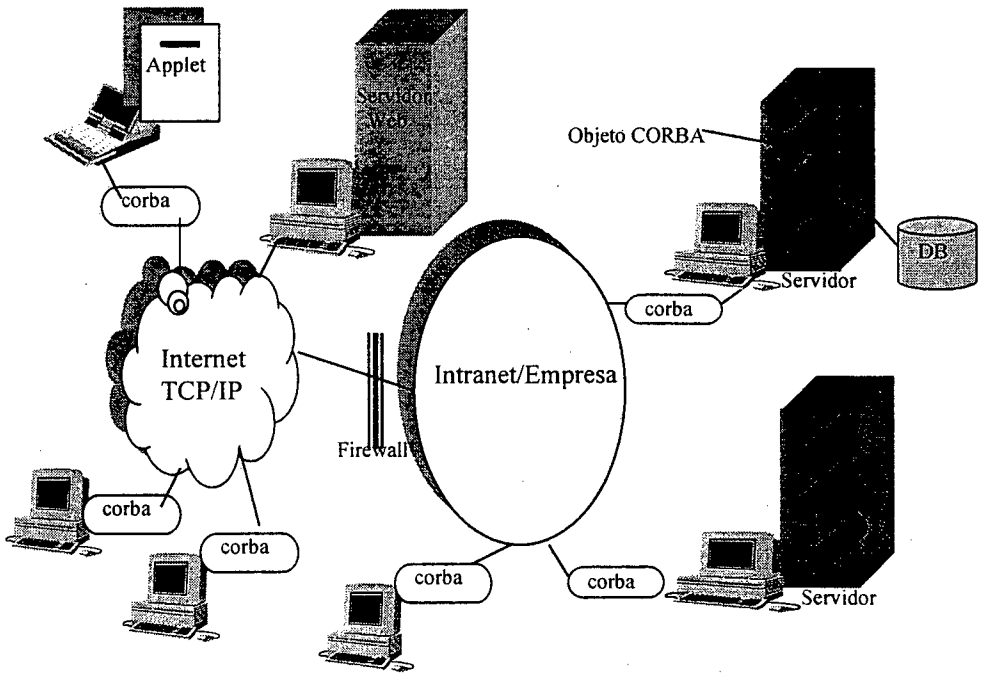

Figura 3.1 - Arquitetura de sistema distribuído usando CORBA e Internet.

[1] Uma camada de software residente acima do sistema operacional que oferece abstrações de alto nível, com objetivo de facilitar a programação distribuíd

Tanto a linguagem quanto o compilador utilizado para a geração do código do objeto-servidor são totalmente transparentes para o cliente. O cliente não precisa saber onde o objeto distribuído está localizado, qual sistema operacional está sendo utilizado para executá-lo ou em que linguagem foi escrito. A única coisa que o cliente necessita saber é a interface de acesso fornecida pelo objeto-servidor [06].

3.3.1 Arquitetura CORBA

Em 1990, o OMG criou o OMA (*Object Management Architecture*) com o objetivo de fornecer uma infra-estrutura conceitual para todas as especificações OMG. O OMA é a arquitetura geral dos componentes necessários para desenvolver um ambiente portável e interoperável. Para ser portável e interoperável, considera a necessidade de interfaces orientadas à objetos, transparência de distribuição, uma forma comum de modelar objetos, uma base comum para todos os componentes, suporte total para todos os estágios do ciclo de vida do software, uma natureza flexível e dinâmica, alto desempenho, implementações robustas, e compatibilidade com padrões existentes dentro da indústria de software [06].

O OMA é composto de quatro elementos principais: *Object Request Broker*, Objetos de Aplicação (*Application Objects*), Serviços de Objetos Comuns (*Corba Services*) e Facilidades Comuns (*Corba Facilities*). A Figura 3.2 apresenta a estrutura geral e oselementos que compõem o OMA.

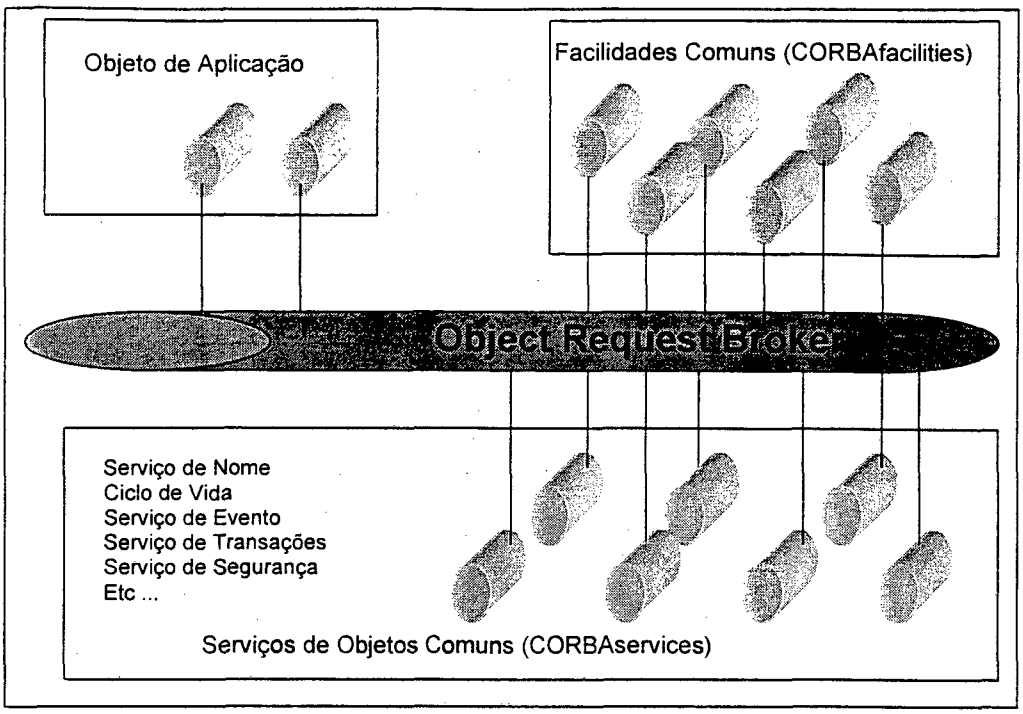

Figura 3.2 - Arquitetura de Gerenciamento de Objetos [06].

Objetos de Aplicação estão relacionados às aplicações específicas do usuário, que são integradas ao ambiente. Essas aplicações não são padronizadas pelo OMG. As Facilidades Comuns propiciam uma coleção de serviços que muitas aplicações podem compartilhar, mas que não são tão fundamentais como os Serviços de Objetos. Os Serviços de Objetos fornecem funções básicas para usar e implementar objetos.

Objetos comunicam-se com outros objetos via ORB (*Object Request Broker*), que é o elemento chave de comunicação. O ORB fornece mecanismos pelos quais os objetos são ativados, enviam requisições e recebem respostas de forma transparente [06].

3.3.2 *Object Request Broker*

Objetos Clientes requisitam serviços das implementações de objetos através de um ORB. O ORB é responsável por todos os mecanismos requisitados para encontrar o objeto-

servidor, preparar a implementação de objeto para receber a requisição, e executar a requisição. A Figura 3.3 ilustra a requisição de uma implementação de objeto.

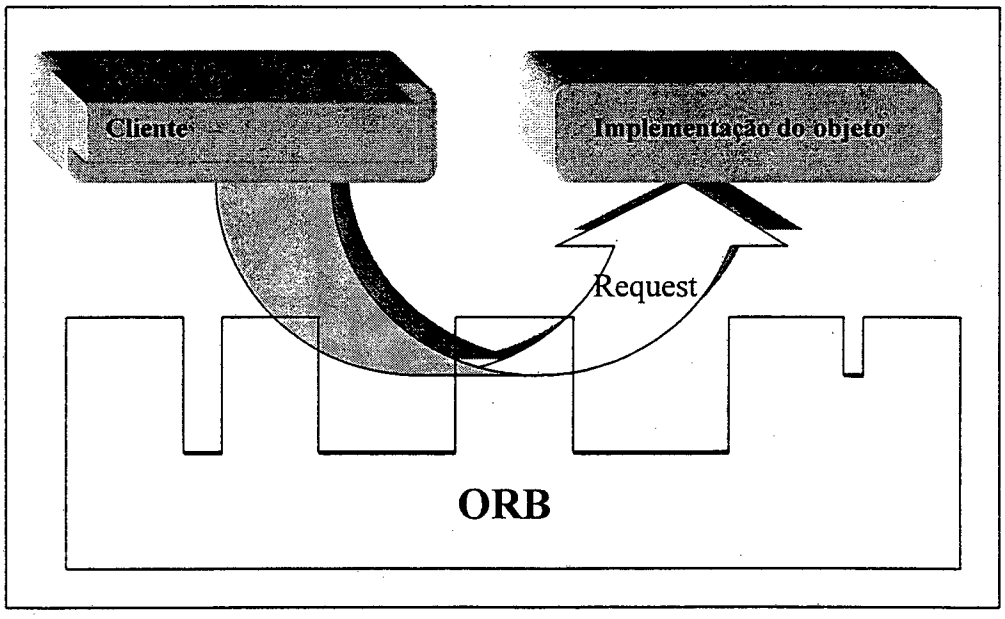

Figura 3.3 - Requisição de uma Implementação de Objeto.

O cliente não tem noção da localização do objeto-servidor, não sabe que linguagem foi utilizada para sua implementação ou qualquer outro detalhe referente à implementação desse objeto. O objeto é definido com o uso da linguagem OMG IDL.

3.3.3 OMG IDL

CORBA utiliza a OMG IDL (*Interface Definition Language*) como forma de descrever interfaces, isto é, de especificar um contrato entre os objetos. A OMG IDL é uma linguagem puramente declarativa. Por definição, todos os serviços OMG devem ser especificados através do uso de uma linguagem declarativa com o objetivo de separar as interfaces das implementações. A OMG IDL não fornece detalhes de implementação. Métodos especificados em OMG IDL podem ser ligados a vários tipos de linguagens, entre elas C, C++, Ada, Smalltalk, COBOL e Java. OMG IDL fornece interface independente de

sistema operacional e linguagem de programação para todos os serviços e componentes que são fornecidos pelo CORBA, permitindo que clientes e servidores desenvolvidos em diferentes linguagens e sistemas operacionais interajam.

3.3.4 Arquitetura do ORB

A arquitetura do ORB é composta de três componentes específicos: Interface de cliente, interface de implementação de objetos e ORB *core* ou núcleo do ORB. Veja a Figura 3.4.

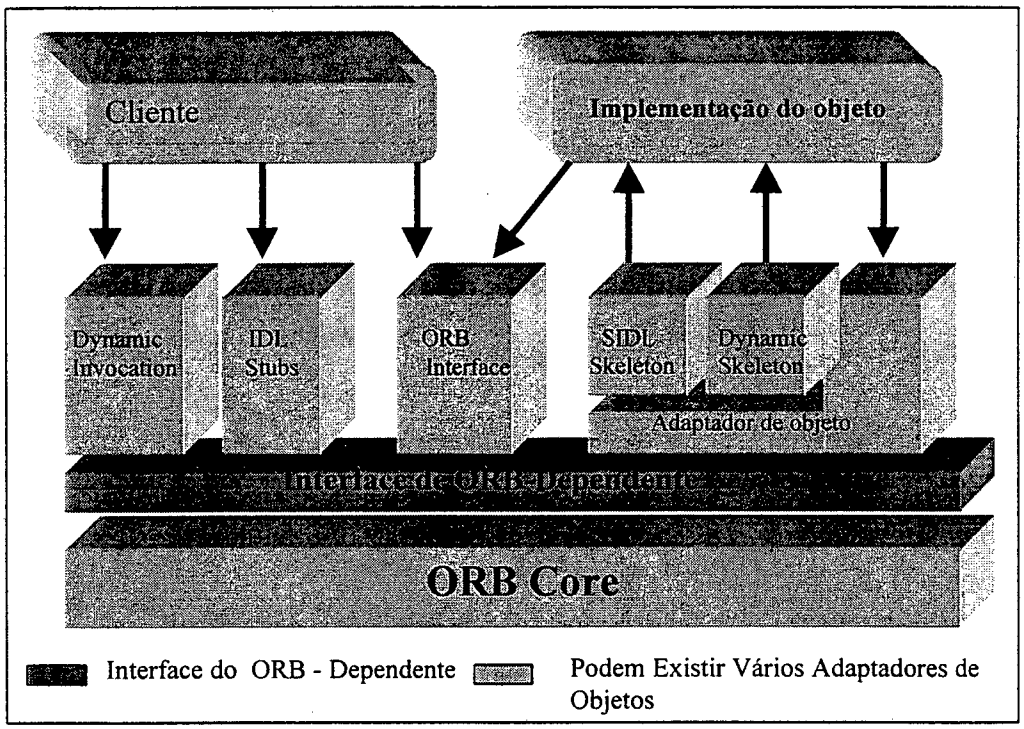

Figura 3.4 - Estrutura das Interfaces de Requisição de Objeto [02].

a) Interface Cliente

A interface de cliente fornece as seguintes interfaces para o ORB e objetos-servidores:

- **IDL *stubs*** – O IDL *stub* representa a interface da composição das funções geradas pela definição da interface IDL e ligadas ao programa cliente. Esta interface de invocação estática representa o mapeamento entre a linguagem cliente e a implementação ORB. A interface stub IDL traz o ORB direto para o domínio da aplicação do cliente.O cliente interage com o servidor remoto invocando suas operações exatamente como se invocasse operações em objetos locais;

- ***Dynamic Invocation Interface* (DII)** – O ORB fornece um mecanismo de invocação dinâmica, permitindo às aplicações clientes a construção de uma requisição em tempo de execução. Usando o mecanismo DII, um objeto é acessado por uma chamada ao ORB ou por uma série de chamadas ao ORB no qual o objeto, métodos e parâmetros estão especificados. É responsabilidade do cliente especificar os tipos de parâmetros e retornos esperados;

- **Repositório de Interface** – Contém a descrição de todos os componentes de interface codificados com a OMG IDL, incluindo as constantes, tipos de dados (*typedefs*), as assinaturas de operações dentre outros. A assinatura para uma operação inclui os argumentos que fazem parte dela, os tipos de dados desses argumentos e o tipo de dado que a operação retorna, caso ela retorne dado. O cliente usa o repositório de interfaces durante uma invocação dinâmica para obter a assinatura da operação solicitada;

Interface ORB – A interface ORB permite que o código do cliente acesse diretamente as funções do ORB. Esta interface fornece somente umas poucas operações, tais como transformar em texto uma referência a objeto. A interface ORB é um dos componentes que é compartilhado pelo lado cliente e pelo lado da arquitetura da implementação. Veja a Figura 3.4.

b) Interface Servidora

A interface de implementação é composta dos seguintes elementos:

- **IDL *skeleton*** – É a parte do lado servidor correspondente ao stub da interface IDL;

- ***Dynamic Skeleton Interface* (DSI)** – A DII especificada no CORBA 1.2 é um mecanismo do lado cliente, não existindo correspondente no lado servidor. Já no CORBA 2, o *Dynamic Skeleton Interface* (DSI) é requerido pela interoperabilidade do ORB. O DSI é o lado servidor correspondente ao DII, provendo um mecanismo de ligação para os servidores entregarem requisições ao ORB para uma implementação cliente que não tenha o conhecimento em tempo de compilação do tipo do objeto. O DSI pode receber tanto invocações dinâmicas quanto estáticas;

- ***Object Adapter*** – Como um servidor CORBA interage com milhões de pequenos objetos? Quem gerencia as referências aos objetos? Como estas referências são armazenadas de forma persistente? A resposta a estas questões é o *Object Adapter*. O *Object Adapter* está no topo dos serviços de comunicação do ORB, aceitando requisições para serviços de interesse dos objetos-servidores; ele fornece o ambiente para instanciar objetos, atribui referências de objetos, e passar requisições a eles. Com um adaptador de objetos, é possível a uma implementação de objeto ter acesso a um serviço independentemente de estar implementado no núcleo do ORB. Se o núcleo do ORB oferecer o serviço, o adaptador simplesmente fornece uma interface para ele; caso contrário, o adaptador deve implementá-lo no topo do núcleo do ORB.

  O conceito de adaptador de objeto é necessário para permitir que o CORBA suporte uma grande variedade de aplicações, com variados tipos e estilos de implementações de objetos. Se os serviços do adaptador de objetos fossem oferecidos diretamente pelo núcleo do ORB, seria necessária a existência de uma gama muito grande de métodos na interface do ORB, para atender todas as demandas dos variados tipos de objetos-servidores existentes.

  Não é necessário que todos adaptadores de objetos ofereçam a mesma interface ou funcionalidade. Algumas implementações de objetos podem ter requisitos especiais,

exigindo adaptador de objetos específicos. Dessa forma, a implementação do objeto pode escolher qual adaptador de objetos utilizar, dependendo de quais tipos de servidores ele requer. Entretanto, com o objetivo de tentar conter uma proliferação interminável de tipos de adaptadores de objetos, [01] recentemente definiu o Adaptador de Objetos Portável o POA (*Portable Object Adapter*). A especificação do POA define um adaptador de objeto que pode ser usado pela maioria dos objetos que possuem implementações convencionais. Ele fornece as seguintes funções: associar uma interface de objetos com sua implementação; gerar e interpretar as referências de objetos e mapeá-las para as implementações de objetos; ativar e desativar objetos e implementações; registrar implementações; invocar implicitamente métodos através do *skeleton*.

Repositório de Implementações – provê um repositório de informações sobre as classes suportadas por um servidor, objetos instanciados e seus identificadores. Veja a Figura 3.5. O Repositório de Implementações é também usado para armazenar informações associadas com implementações de ORBs

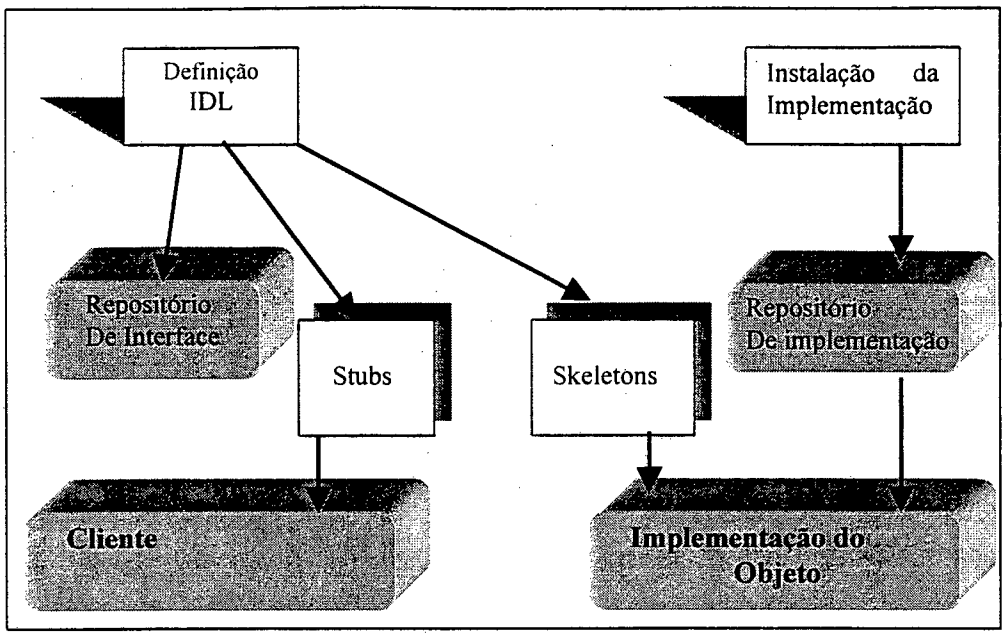

Figura 3.5 - Repositório de Interface e Implementação [02].

O CORBA usa Repositório de Implementações para associar as referências de objetos com suas implementações. Muitas informações no Repositório de Implementações são próprias aos fabricantes que determinam quais operações usar para construir e acessar o repositório;

- **Interface ORB** – Esta interface é compartilhada pelas implementações dos objetos e pelos objetos clientes.

c) Núcleo de ORB

O núcleo de ORB ou ORB Core é responsável pela manipulação da comunicação básica das requisições dos vários componentes, podendo ser compreendido como uma camada básica de transporte, como pode ser visto na Figura 3.5.

3.4 Interoperabilidade entre ORBs

A especificação de interfaces de objetos (obrigatoriamente em OMG IDL), garante a portabilidade dos objetos através de diferentes linguagens, ferramentas, sistemas

operacionais e redes. Entretanto, a característica de interoperabilidade de objetos só foi possível no CORBA versão 2. Isso se deu através da especificação de uma arquitetura de interoperabilidade, um suporte para pontes entre ORBs, um protocolo para comunicar entre ORBs genéricos e um protocolo para comunicação entre ORBs para *Inernet.* A arquitetura de interoperabilidade do CORBA identifica claramente a regra de diferentes tipos de domínios para informações específicas de ORBs. Tais domínios podem incluir domínios de referências de objeto, domínios de tipos, domínio de segurança, dentre outros. Quando dois ORBs estão no mesmo domínio, eles podem se comunicar diretamente. Entretanto, quando a informação em uma invocação deve deixar seu domínio, a invocação deve atravessar uma ponte.

A regra de ponte deve assegurar que o conteúdo e a semântica sejam mapeados adequadamente de um ORB para outro. O suporte para ponte entre ORBs pode também ser usado para prover interoperabilidade com outros sistemas não-CORBA. Uma ponte que provê uma conversão direta é chamada de *immediate bridging.* Uma ponte que requer um formato de interoperabilidade para ser usado externamente para comunicação entre dois ambientes de ORB é chamada de *mediated bridging.* Para a comunicação entre ORBs o OMG definiu os protocolos GIOP e IIOP.

3.4.1 GIOP e IIOP

O GIOP (*Generic Inter-ORB protocol*) especifica uma sintaxe de transferência padrão e um conjunto de formatos de mensagens para comunicação entre ORBs. O IIOP (*Internet Inter_ORB Protocol*) especifica como mensagens GIOP são trocadas usando conexões TCP/IP.

O relacionamento entre GIOP e IIOP é similar ao mapeamento existente entre o OMG IDL e uma linguagem especifica: o GIOP pode ser mapeado em diferentes protocolos de transporte, e especifica os elementos de protocolo que são comuns a todos os mapeamentos. Entretanto, o GIOP não fornece uma completa interoperabilidade, da mesma forma que

IDL não pode ser usado para se construir programas completos. IIOP, e outros mapeamentos similares para diferentes protocolos de transporte, são realizações concretas das definições abstratas de GIOP. Veja a Figura 3.6

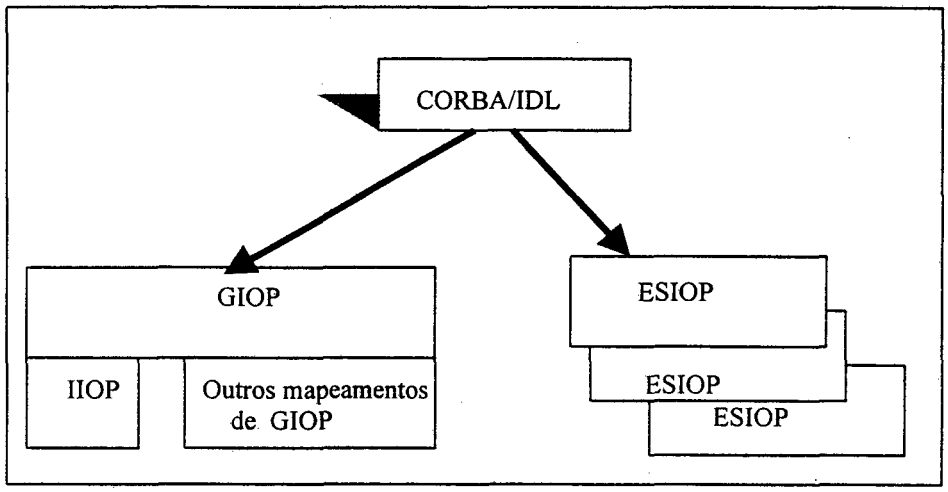

Figura 3.6 Estrutura do Protocolo IIOP.

O ESIOP (*Environment Specific Inter ORB Protocol*) protocolo de comunicação entre ORBs para ambientes específicos deve ser usado para interoperar em locais onde uma rede ou infra-estrutura de computação distribuída já esteja em uso. Apesar de cada ESIOP poder ser otimizado para um arquitetura em particular, toda especificação ESIOP deve estar conforme as convenções da arquitetura de interoperabilidade para facilitar o uso de pontes, se necessário. O suporte de ponte entre ORBs habilita às pontes serem construídas entre domínios de ORBs que usam IIOP e domínios que usam um ESIOP particular.

3.5 Serviços CORBA

O ORB, por si só, não executa todas as tarefas necessárias para que os objetos interoperem. Ele só fornece os mecanismos básicos. Outros serviços necessários são oferecidos por objetos com interface IDL, que o OMG vem padronizando. Esses serviços são utilizados

para aumentar e complementar a funcionalidade do ORB. Até a data da elaboração deste trabalho foram publicados 15 serviços, entre esses estão:

- ***Naming Service*** – permite a componentes da rede localizar outros componentes pelo nome. Este serviço também permite a localização de objetos através de diretórios de rede ou outros tipos de serviços de nomes;

- ***Event Service*** – permite a componentes da rede dinamicamente registrarem ou não seu interesse em um evento específico. Este serviço define um objeto chamado *event channel* que coleta e distribui eventos entre os componentes que não sabem nada uns dos outros;

- ***Security Service*** – provê um framework completo para segurança de objetos distribuídos.

3.6 Conclusão

CORBA é uma arquitetura aberta que associa o paradigma cliente/servidor com a tecnologia de orientação a objetos. Ela permite a integração de aplicações distribuídas heterogêneas, fazendo com que aplicações façam solicitações a objetos, de uma forma transparente e independente, indiferente às plataformas de hardware, sistemas operacionais, linguagens e considerações de localização.

O modelo CORBA oferece uma série de benefícios que facilitam a integração de aplicações em um ambiente distribuído. Este modelo permite que os projetistas de sistemas distribuídos tirem proveito de técnicas orientadas a objeto como encapsulamento e herança. Estas técnicas fazem com que os objetos sejam definidos como "caixas pretas" capazes de executar determinadas tarefas, sem que o sistema necessite conhecer o funcionamento interno destes objetos. Outra vantagem é a definição mais clara das interfaces entre as partes do sistema, as quais podem ser alteradas sem afetar o sistema como um todo. Assim,

consegue-se uma melhor modularidade. Os sistemas tornam-se mais extensíveis pois novas características podem ser adicionadas através da definição de um novo conjunto de operações. CORBA pode coexistir e interoperar perfeitamente com outros sistemas que não foram desenvolvidos usando a tecnologia CORBA, permitindo assim preservar o investimento do sistema existente. Por exemplo, um objeto CORBA pode se comunicar com um objeto desenvolvido com a tecnologia ActiveX/DCOM da Microsoft. CORBA pela sua arquitectura aberta, sua flexibilidade, sua interoperabilidade e sua portabilidade pode ser visto como a solução dos problemas encontrados em sistemas distribuídos. É importante salientar que CORBA é um padrão cada vez mais utilizado na área de gerência de redes [33].

Essas vantagens justificam a motivação da escolha desta arquitetura para o desenvolvimento deste trabalho.

O próximo capítulo apresenta as tecnologias da *Internet* pois um dos objetivos neste trabalho é integrar as tecnologias da *Internet* com a arquitetura CORBA, mostrando que esta integração é perfeita para construção de aplicações distribuídas heterogêneas.

Capítulo 4

Tecnologias Chaves da Internet

4.1 Introdução

Enquanto CORBA 2.0 estavam sendo especificado a *Internet* e o *World Wide Web* se tornavam um fenômeno explosivo de rápida expansão, e até hoje continua crescendo. A *Internet* ultrapassou os ambientes de órgãos do governo e de instituições de educação que lhe foram impostas no início da sua criação, para ser hoje o meio mais significativo para a comunicação entre empresas, organizações governamentais educacionais e de pessoas físicas. Através da *Internet* pode–se acessar, fornecer e trocar um conjunto quase ilimitado de informações de uso empresarial, acadêmico e pessoal. O crescimento da Internet e das *intranets* continuará a grande velocidade. A *Internet* é uma interconexão global sem precedente na história da computação [07]. Desenvolvimentos relacionados à *Internet* desempenham funções chaves que permitem às organizações de Tecnologia de Informações criar e utilizar melhor os objetos distribuídos. Estes desenvolvimentos também chamados de tecnologias da Internet são: *Java*, *Applet Java*, *Web Browser* [31].

4.2 Java

Uma poderosa linguagem de programação orientada a objetos desenvolvida pela *Sun MicroSystems*. Embora seja conhecida como uma linguagem para o *World Wide Web*, *Java* pode ser utilizado para desenvolver aplicações simples e complexas. Programas *Java* são

compilados e geram o *byte-code* que é um código que é executado na Maquina Virtual (*Virtual Machine*) Java, permitindo que todo programa *Java* seja executado independentemente do processador e do sistema operacional. Isso significa um avanço sobre as outras linguagens de programação. Em aplicações distribuídas isso é muito importante na medida em que uma aplicação cliente ou um objeto-servidor pode ser executado em qualquer plataforma. Conseqüentemente o custo de desenvolvimento e de manutenção pode ser reduzido significativamente.

4.3 *Applet Java*

São pequenos programas escritos em Java, móveis do servidor *Web* até o cliente onde eles são executados no contexto de um *browser Web*. Uma das vantagens dos *applets* é que eles permitem evitar o problema de incompatibilidade de versão que existe com os softwares aplicativos onde muitas vezes são carregados e atualizados em cada máquina cliente.

4.4 *Web Browsers*

São navegadores de páginas de hipermídia como por exemplo *Netscape* e *Internet Explorer*. Esses navegadores são fornecidos gratuitamente e vêm em milhões de computadores. Eles permitem visualizar as informações apresentadas na *Web* e incluem suporte para os serviços da *Internet*. Eles também possibilitam executar os *Applets Java* que podem se comunicar e interagir com aplicações executadas em sistemas remotos via a rede *Internet*. Desta forma, as empresas de Tecnologia de Informações estão usando a *Internet* de fato como um *WAN* que interliga agências do governo, empresas de negócios, instituições de educação e indivíduos.

4.5 Segurança dos *Firewalls*

Os *firewalls* (muros de pedras contra fogo) fornecem uma segurança para os sistemas de informações das empresas permitindo que os aplicativos internos da empresa interajam de maneira segura com os aplicativos em sistemas fora do *firewall*. Numa rede, o *firewall* é um sistema que controla e permite acesso de fora para dentro (e vice-versa) somente aos usuários autorizados, evitando acessos indevidos, sejam esses acessos de usuários internos da rede, tentando acessar informações ou sites não autorizados na *Internet*, ou acessos de usuários externos (usuários maliciosos) vindo de fora pela *Internet* para sua rede interna.

4.6 Conclusão

As atuais revoluções na *Internet* e na computação de objetos distribuídos convergem no mesmo caminho. A *Internet,* vista como um *framework* de comunicações, fornece uma plataforma ideal para aplicação de objetos distribuídos. Ao mesmo tempo, a tecnologia de objetos distribuídos está melhorando a qualidade das aplicações baseadas em *Web*, dando mais valores a *Internet* e *Intranet* nas empresas de Tecnologia de Informações. Esse relacionamento simbiótico entre as duas tecnologias cria um novo modo de conceituar, projetar, desenvolver, executar e manter as aplicações de negócios. Isso faz surgir um novo paradigma de aplicação, construída a partir de códigos fontes encapsulados na forma de objeto executado em sistema protegido, ou não, por *firewall* e interagindo uma com a outra através de um ORB. Este novo paradigma de aplicação tem muitos benefícios a oferecer às empresas de Tecnologia de Informações, como por exemplo, flexibilidade para integrar aplicações de diferentes fabricantes, administração e gerenciamento centralizados dos objetos aplicativos, drástica redução no custo da configuração e manutenção dos sistemas clientes, dentre outros.

As empresas que empregam ou se adaptam a estes novos conceitos sem dúvida terão uma grande vantagem sobre seus competidores.

Na era da Internet não se pode mais pensar em criar ambientes de computação homogêneos, os sistemas de informações das empresas devem ser capazes de se comunicar e de interoperar com aplicações e sistemas fora do *firewall* da empresa e em ambientes heterogêneos. A Internet é uma interconexão de redes em nível mundial em rápida expansão, com mais de dois milhões de organizações acadêmicas, militares, científicas e comerciais. Ela oferece inúmeros serviços. Porém nosso objetivo neste trabalho é focalizar os aspectos que, junto com a arquitetura CORBA permitem desenvolver aplicações heterogêneas capazes de proporcionar uma melhor forma para a integração no processamento das informações distribuídas, ou seja, aplicações que suportam interoperabilidade.

Em sistemas distribuídos, entretanto, as informações são vulneráveis a ataques de usuários maliciosos. O próximo capítulo apresenta os serviços de segurança, que consistem em medidas que permitem deter, prever, detectar e corrigir violações que envolvem a transmissão de informações.

Capítulo 5

Segurança em Sistemas de Informação

5.1 Introdução

Nas últimas décadas, os requisitos da segurança de informações nas organizações sofreram duas mudanças [26]:

A primeira delas foi a introdução dos computadores. Então, a necessidade de ferramentas automatizadas para proteger os arquivos e as outras informações armazenadas nos computadores se tornou evidente. Tais são os casos de sistemas compartilhados e sistemas que podem ser acessados via telefones públicos (acessos discados) e redes de dados. O nome genérico dado para essa segurança de informações é **Segurança de Computador**.

A segunda mudança que afetou a segurança foi a introdução dos sistemas distribuídos e o uso de redes e facilidades de comunicação para transportar dados entre o terminal de usuário e o computador, e entre computador e computador. Medidas de **Segurança de Rede** são necessárias para proteger os dados durante sua transmissão.

Não existe um limite claro entre as duas formas de segurança.(**Segurança de Computador e Segurança de Rede**) Por exemplo, um dos mais conhecidos tipos de ataque em sistema de informação é o vírus de computador. O vírus pode ser introduzido no sistema fisicamente por um disquete ou pode chegar por uma rede. Em ambos os casos, quando o

vírus está no sistema do computador, ferramentas internas de segurança são necessárias para detectá-los e removê-los.

Neste trabalho focaliza-se a Segurança de Rede que consiste em medidas que permitem deter, prever, detectar e corrigir violações que envolvem a transmissão de informações.
Neste capítulo discutem-se os diferentes tipos de ataques que criam uma necessidade de mecanismos e serviços de Segurança de Rede.

5.2 Ataques à Segurança

Ataques à segurança de um sistema de computador ou de rede são mais caracterizados pelas funções do sistema de computador como o fornecimento de informações. Em geral, há um fluxo de informações da origem (arquivo ou região de memória principal) para o destino (um outro arquivo ou usuário). Tipos de ataques :

- **Interrupção:**

Um componente ativo do sistema é destruído, se torna indisponível ou inutilizável. Isso é o ataque à **disponibilidade,** por exemplo, destruição de peça de hardware como o disco rígido, corte de linha de comunicação, ou inabilidade do sistema de gerenciamento de arquivos.

- **Interceptação**:

Uma parte não-autorizada ganha o acesso de um componente. Isso é um ataque à **confidencialidade**. A parte não autorizada pode ser uma pessoa, um programa ou um computador. Por exemplo, uma conexão secreta para capturar dados na rede e copiar de maneira ilícita arquivos ou programas.

- **Modificação**:

Uma parte não-autorizada não somente tem acesso, mas também altera, falsifica o componente. Isso é um ataque à **integridade**. Por exemplo, modifica valores no arquivo de dados, altera um programa para ter comportamento diferente daquele previsto, e modifica o conteúdo de uma mensagem que está sendo transmitida.

- **Fabricação**:

Uma parte não-autorizada insere um objeto forjado ou falsificado no sistema. Isso é um **ataque à autenticidade**. Por exemplo, inserir mensagens ilegítimas em uma rede ou adicionar registro em um arquivo. A Figura 6.1 ilustra os quatro tipos de ataque à segurança de informação.

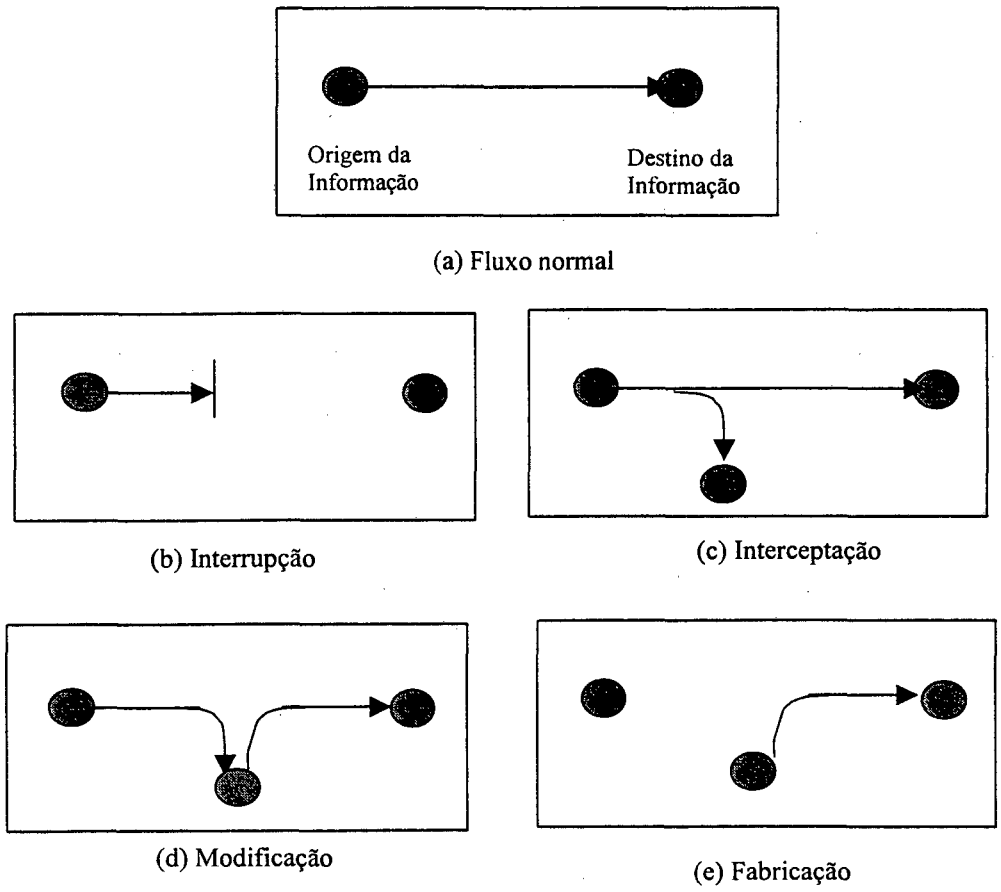

Figura 5.1 Ataques à Segurança de Informação [26].

Os ataques à segurança podem ser agrupados em duas categorias: ataque passivo e ataque ativo. A Figura 5.2 apresenta as duas categorias [26].

5.2.1 Ataque Passivo

No ataque passivo o objetivo do intruso é interceptar a informação que está sendo transmitida. Existem dois tipos de ataque passivo que são: exibir o conteúdo da mensagem e analisar o tráfego [26].

Exibir o conteúdo é fácil de entender. Por exemplo, o intruso escuta uma conversa telefônica, fica sabendo do conteúdo da mensagem de correio eletrônico ou de arquivo sendo transmitido. O segundo tipo de ataque passivo é a **análise de tráfego**. Esse ataque é mais sutil. Supondo uma mensagem criptografada em transmissão, neste caso o intruso não teria acesso ao seu conteúdo, mas poderia observar o caminho da mensagem, definir a localização, a identidade das máquinas (*host*) de comunicação, observar a freqüência e o tamanho da mensagem que está sendo transmitida. Estas informações podem ser importantes para deduzir a natureza da comunicação que está acontecendo.

O ataque passivo é muito difícil de detectar, pois ele não envolve a alteração de dados, assim o seu tratamento consiste mais em prever do que detectar.

5. 2.2 Ataques Ativos

Estes ataques envolvem modificações de dado ou a criação de falso dado. Eles podem ser subdivididos em quatro categorias: mascarado, *replay*, modificação de mensagens e negação de serviço.

O **Mascarado** acontece quando uma entidade pretende ser diferente. Um ataque mascarado geralmente inclui uma das outras formas de ataque ativo. Por exemplo, uma seqüência de autenticação pode ser capturada e reutilizada depois que ocorreu a validação, capacitando assim uma entidade autorizada de pouco privilégio a obter um privilégio maior.

Replay envolve uma captura passiva de uma unidade de dado e sua retransmissão subseqüente para produzir um efeito não autorizado.

Modificação de mensagens simplesmente significa alterar uma porção de uma mensagem legítima, atrasá-la ou reordenar a seqüência para produzir um efeito não autorizado. Por exemplo uma mensagem " Autorizo WC a retirar R$ 100.000,00 da conta 20-0000" é modificado para ter um outro significado " Autorizo DN a retirar R$ 100.000,00 da conta 20-0000".

A negação de serviço impede ou inibe o uso normal ou o gerenciamento das facilidades de comunicação. Este ataque pode ter um alvo específico; por exemplo, uma entidade pode eliminar todas as mensagens direcionadas a um destino particular (serviço de auditoria de segurança). Outra forma de serviço negado é o transtorno de uma rede inteira, desabilitando ou sobrecarregando a rede de mensagens à fim de degradar a sua performance.

Os ataques ativos apresentam características opostas aos passivos, sendo que os passivos são difíceis de detectar. Entretanto, existem medidas para prever seu acontecimento (sucesso). De outro lado, os ataques ativos são muito difíceis de prever. Para fazer isso requer-se a todo instante uma proteção física das facilidades de comunicação e dos caminhos das mensagens. O objetivo é detectá-los, se recuperar de sua ação ou refazer qualquer transtorno ou atraso que eles causarem. Detectar tem um efeito dissuasivo e pode também contribuir com a prevenção.

Para evitar essas ataques deve-se empregar serviços de segurança na definição de uma política de segurança.

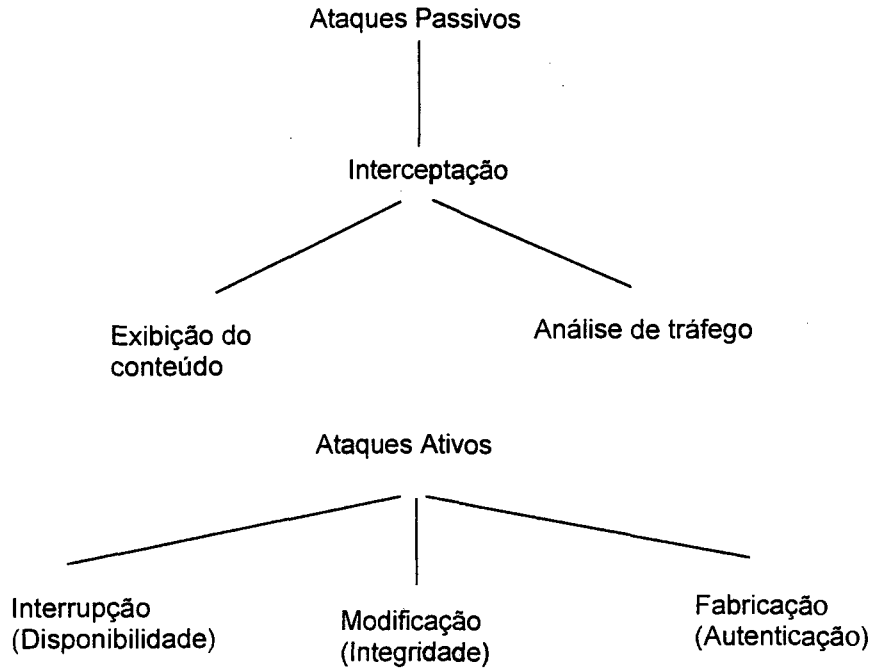

Figura 5.2 Ataques passivos e Ativos a Segurança de Rede [26].

5.3 Serviços de segurança

Os serviços de segurança consistem em medidas que permitem deter, prever, detectar e corrigir violações que envolvem a transmissão de informações. Isso envolve :

5.3.1 Confidencialidade

O serviço de confidencialidade permite proteger dados contra ataques passivos. Esse serviço pode ser total ou parcial. O serviço total é conhecido como "*Broadest Service*". Por exemplo, quando um circuito virtual é estabelecido entre dois sistemas para transmissão de dados, o serviço total impede que um intruso exiba as mensagens que estão sendo

transmitidas. O serviço parcial é um refinamento do serviço total, ele protege uma única mensagem ou um campo específico da mensagem. Esse último serviço é difícil de implementar e pouco usado.

Um outro aspecto da confidencialidade é a proteção do fluxo de tráfego impedindo o intruso de analisar o tráfego, ou seja, impedir que ele observe a origem, o destino, a freqüência das mensagens, e outras características do tráfego.

5.3.2 Autenticação

O serviço de autenticação garante a autenticidade da comunicação. No caso de uma única mensagem como por exemplo um sinal de alarme, a função do serviço de autenticação é garantir que o emissor seja realmente quem ele diz ser. No caso de uma iteração de mensagens como por exemplo a conexão de um terminal à uma estação, dois aspectos são considerados. No início da conexão, o serviço deve garantir que as duas entidades (emissora e o receptora) sejam realmente quem elas dizem ser. O segundo aspecto, é que o serviço deve assegurar que uma terceira entidade indesejável não esteja, se mascarando e se fazendo passar por uma das duas entidades legítimas.

5.3.3 Integridade

Como o serviço de confidencialidade, o serviço de Integridade pode ser total ou parcial. Esse serviço garante que as mensagens sejam recebidas do jeito que elas foram enviadas sem duplicação, alteração, reordenação e atraso. A recuperação de dados destruídos também pode ser tratada pelo serviço de integridade. Quando o serviço é parcial, geralmente ele protege somente as mensagens contra a alteração. O serviço de integridade trata dos ataques ativos, conseqüentemente, ele detecta, mas não previne contra ataques. Quando uma violação da integridade é detectada o serviço notifica ou relata a ocorrência. Uma outra parte de software ou uma intervenção humana é requerida para tratar a violação. Alternativamente, existe mecanismo de serviço de integridade disponível que recupera a perda de dados. Esta alternativa é mais atrativa.

5.3.4 Não-Repúdio

O serviço de não-repúdio impede que a entidade emissora e receptora neguem respectivamente a emissão e a recepção de mensagens. Quando uma mensagem é enviada, a receptora pode provar que a mensagem foi, de fato, enviada por uma emissora declarada. Similarmente, quando uma mensagem é recebida, a emissora pode reciprocamente provar que ela foi recebida por uma receptora declarada.

5.3.5 Controle de Acesso

No contexto de segurança de rede, o controle de acesso é a habilidade de limitar e controlar o acesso a sistemas de estações, arquivos e objetos via link de comunicação. Para efetuar esse controle, cada entidade que tente acessar um desses elementos deve ser primeiramente autenticada.

5.3.6 Disponibilidade

Ataques a segurança podem resultar em perda total ou redução da disponibilidade dos elementos do sistema distribuído. Alguns desses ataques são corrigidos por medidas como a autenticação e a criptografia. Mas outros requerem ações físicas.

5.4 Conclusão

Neste capítulo foram discutidos os diferentes tipos de ataques que criam uma necessidade de mecanismos e serviços de Segurança de Rede. Foram Apresentados sucintamente esses serviços que consistem em medidas de Segurança que permitem deter, prever, detectar e corrigir violações que envolvam a transmissão de informações. O próximo capítulo resume a definição do modelo de referência de segurança segundo

o OMG. Este modelo permitirá o melhor entendimento do mecanismo de segurança proposto neste trabalho.

Capítulo 6

Modelo de Segurança em Sistema CORBA

6.1 Introdução

O modelo de segurança em um sistema CORBA usa, de acordo com as especificações do OMG, as noções de clientes, objeto-destino, e de invocações de operações. Os processos de preparo da invocação, a transmissão da invocação para o objeto-destino através da rede, a execução da operação e a entrega da resposta são reforçados por procedimentos de segurança que respeitam as políticas de segurança definidas nos domínios.

Os domínios são escopos onde características e regras comuns são exibidas e observadas sobre um conjunto de objetos a fim de garantir a segurança de acesso a esses objetos.

Este modelo é muito geral. O objetivo é incluir todas as possíveis funções de segurança que uma aplicação pode requerer. Mas isso não significa que todas as funcionalidades descritas podem ser suportadas por todas as tecnologias que implementam a segurança. A visão apresentada neste capítulo não é exaustiva, mas está dentro do enfoque desejado para a evolução deste trabalho.

6.2 Definição de um Modelo de Referência de Segurança

Um modelo de referência descreve como e onde um sistema seguro pode encontrar políticas de segurança. Essas políticas de segurança definem:

- As condições para que as entidades ativas, como clientes que agem em nome de usuários, possam acessar objetos;

- A autenticação e a autorização de usuários e outros *principals* para provar quem eles dizem ser, o que eles podem fazer, e se eles podem delegar seus direitos a outros usuários. Um *principal* é um usuário humano ou entidade de sistema registrado e autêntico [01];

- A segurança de comunicações entre objetos e a qualidade de proteção dos dados em trânsito;

- As responsabilidades das atividades de segurança relevantes que são necessárias.

Um modelo de referência de segurança, em resumo, define um conjunto específico de políticas de segurança. Em outras palavras é uma meta-política. Esta meta-política pretende cercar todas as possíveis políticas de segurança apoiadas pelo OMA(Object Management Architecture). Ela define as interfaces abstratas que são providas pela arquitetura de segurança. As seções a seguir descrevem os elementos do modelo.

6.3 *Principals* e seus Atributos de Segurança

Em um modelo de segurança, os usuários humanos e as entidades internas do sistema são chamados de *principals* e têm suas identidades autenticadas. Às vezes, o *principal* pode ter

mais de uma identidade. Por exemplo, um usuário pode ter uma identidade de acesso que lhe permite entrar no sistema e uma identidade de auditoria que é somente conhecido pelo administrador do sistema A autenticação do *principal* é para ter um nome seguro. Este nome é usado junto com outras informações de autenticação como por exemplo *password* para produzir os atributos de privilégios. Em resumo, os atributos da credencial são importantes para:

- Tornar o *principal* responsável por suas ações;
- Obter acesso aos objetos protegidos;
- Identificar o originador de uma mensagem;
- Identificar a quem cobrar pelo uso do sistema, quando for este o caso.

Estes atributos são divididos em três categorias: atributos não autenticados ou públicos, atributos autenticados ou privados e atributos obtidos por delegação. Todos esses atributos são mantidos em uma credencial como mostra a Figura 6.1.

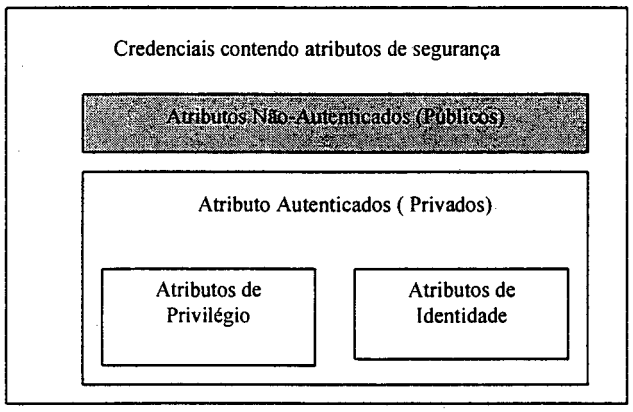

Figura 6.1 - Uma Credencial.

6.4 Invocações Seguras de Objetos Distribuídos

Uma invocação segura é feita usando os seguintes passos:

- Estabelecer uma associação segura entre o cliente e o objeto destino. Isso significa ter um nível de confiança satisfatório nas identidades do cliente e do objeto-destino. O cliente deve sempre ter a segurança de receber referência segura e confiável do objeto de destino e somente o objeto-destino deve executar a autenticação da identidade do cliente;

- O serviço de segurança decide, baseado na identidade autenticada e nos privilégios do cliente se a invocação requisitada é permitida;

- Se a política de segurança prevê uma auditoria, então, neste caso, as informações sobre a invocação são enviadas a um sistema de auditoria. Isso pode ser feito pela aplicação ou automaticamente pelo serviço de segurança baseando-se na política definida pelo administrador de segurança; e

- Dependendo da qualidade de proteção requerida. tanto pelo cliente como pelo objeto destino, a mensagem que transita entre os dois pode ser protegida contra a espionagem e contra qualquer tipo de modificação.

6.4.1 Delegação de Privilégios

O objeto-destino pode ser cliente de um outro objeto. Neste caso, o objeto intermediário quando fizer uma invocação precisa dos privilégios do cliente. Para facilitar isso, o sistema de segurança pode permitir que os privilégios de um *principal* sejam delegados a um outro *principal*. Os esquemas de delegação de privilégios são:

- Delegação simples;

O cliente permite ao objeto intermediário delegar seus privilégios para um outro objeto, o objeto-destino recebe somente os privilégios do cliente e não sabe quem é o objeto intermediário;

Figura 6.2 - Delegação Simples.

- Delegação composta;

O cliente permite ao objeto intermediário delegar seus privilégios para um outro objeto, ambos os privilégios do cliente e do intermediário são passados para o objeto destino. Neste caso o objeto destino sabe quem são o cliente e o intermediário; e

Figura 6.3 - Delegação Composta.

- Delegação de privilégios combinados

O cliente permite que o objeto intermediário delegue seus privilégios para um outro objeto, o intermediário junta os dois tipos de privilégios num só, criando uma única credencial. Neste caso, o objeto-destino não consegue distinguir a origem dos privilégios.

Figura 6.4 - Delegação de Privilégios Combinados.

6.5 Serviços de Não-Repúdio

Os serviços de não-repúdio atuam no sentido de tornar os usuários e outros *principals* responsáveis pelas suas ações. Quando os *principals* executam ações e eventos, os serviços de não repúdio geram indícios ou evidências a serem usados mais tarde como provas, desta forma os *principals* tornam-se responsáveis pelas suas ações. Estas ações podem incluir a criação de um novo objeto, de uma solicitação ou uma resposta, etc. Os componentes dos serviços de não-repúdio são:

- Geração e verificação de provas;

- Armazenamento e recuperação de provas; e

- Uma autoridade de distribuição; Esta autoridade é responsável pela entrega de provas aos objetos do sistema. Ela também entrega as provas a um árbitro que é consultado quando há uma disputa sobre uma ação ocorrida no sistema. A Figura 6.5 ilustra um esquema do serviço de não repúdio segundo o OMG.

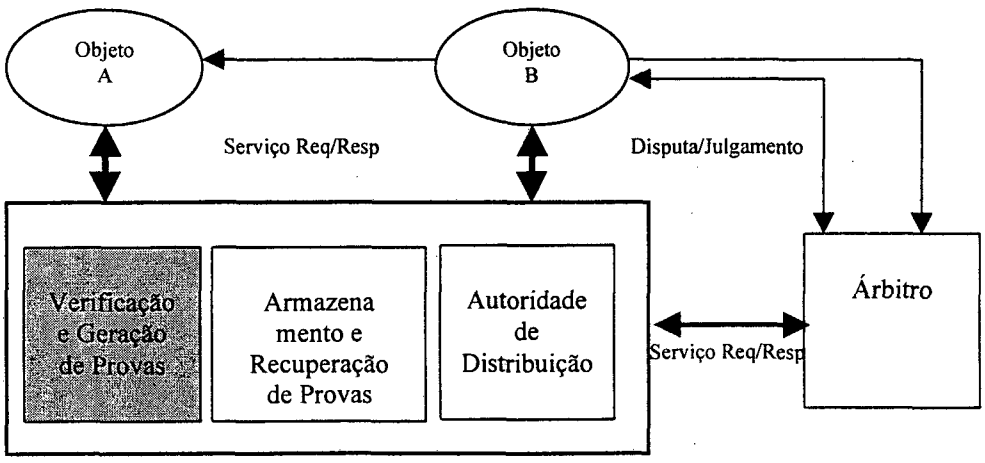

Figura 6.5 - Serviço de Não-Repúdio [01].

6.6 Domínios de Segurança

O domínio de segurança é um escopo onde características e regras comuns são exibidas e observadas sobre um conjunto de objetos a fim de garantir a segurança de acesso aos objetos. Esses objetos, denominados membros do domínio, podem também ser aplicações. O controle de acesso, a autenticação, a invocação segura de objeto e a delegação são orientados por uma política de segurança. O domínio pode ser organizado em uma hierarquia, onde dentro de um domínio podem existir outros domínios, chamados de sub-domínios. Veja a Figura 3.6.

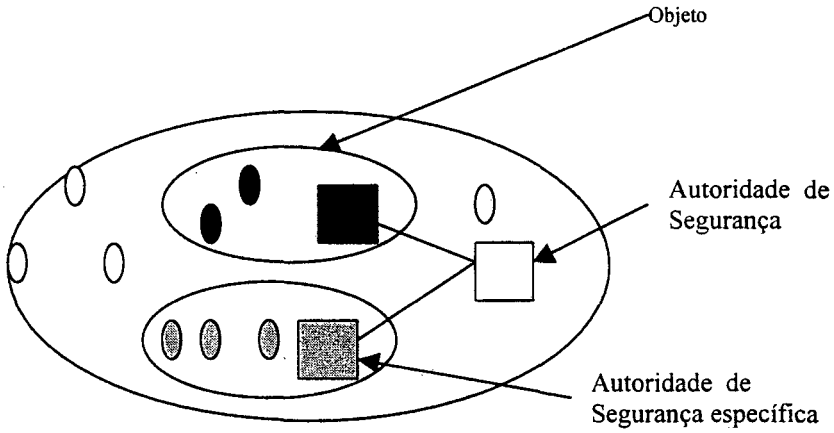

Figura 6.6 - Domínio de Segurança Hierárquico.

6.7 Conclusão

O propósito do modelo de referência especificado pelo OMG é apresentar um modelo que seja flexível. Foram definidas diferentes políticas de segurança que podem ser usadas para alcançar um nível apropriado de funcionalidade e garantir invocação de operação segura entre objetos clientes e servidores. Este modelo de segurança pode ser visto como um guia de referência para a arquitetura de segurança.

O estudo do modelo de referência especificado pelo OMG foi a primeira etapa do desenvolvimento deste trabalho. Neste capítulo, os aspectos referentes à portabilidade,

interoperabilidade, performance e os níveis de segurança não foram tratados. Detalhes podem ser encontrados no [01]. Os conceitos discutidos neste capítulo servem para o melhor entendimento do mecanismo de segurança implementado neste trabalho. O próximo capítulo descreve as características relevantes das ferramentas utilizadas na implementação do projeto.

Capítulo 7

Ferramentas Utilizadas

7.1 Introdução

Este capitulo apresenta as ferramentas utilizadas na implementação do mecanismo proposto. Inicialmente, apresentam-se as características relevantes do *Visibroker;* em seguida, o JDK1.2.2 e, finalmente, o IDE do Visual Café.

7.2 *Visibroker For Java 3.4 da Inprise*

O *Visibroker* For Java da *Inprise* foi escolhido para realizar este trabalho por ser o produto mais usado no mercado [03,08] e mais citado na literatura consultada. Foi utilizada a versão 3.4.

O *Visibroker For Java 3.2* foi a primeira implementação de ORB para Java lançado no mercado. Foi o ORB adotado pela *Netscape* para ser incorporado no seu *browser* conhecido como *Communicator.*

O *Visibroker For Java* vem com um serviço de nomes tolerante a falhas chamado *OSAgent.* Múltiplos OSAgentes em execução na rede localizam-se uns aos outros e definem um espaço de nomes compartilhado entre eles. Isso permite replicar e balancear objetos em várias máquinas. No caso de falha de um objeto, o ORB automaticamente redimensiona a chamada do cliente para uma das réplicas.

O *Visibroker For Java* vem com um serviço *GateKeeper*. O *GateKeeper* permite aos *applets* se comunicarem com os objetos-servidores na rede [32].

O *Visibroker* inclui, na implementação do ORB, os Interceptadores (não especificados pelo OMG). Os Interceptadores são estruturas que permitem interceptar, em certos pontos, as invocações do cliente e as respostas do servidor. Utilizando estes Interceptadores pode-se ver e modificar a comunicação entre o cliente e o servidor, podendo assim alterar o comportamento do ORB. Veja a Figura 8.1.

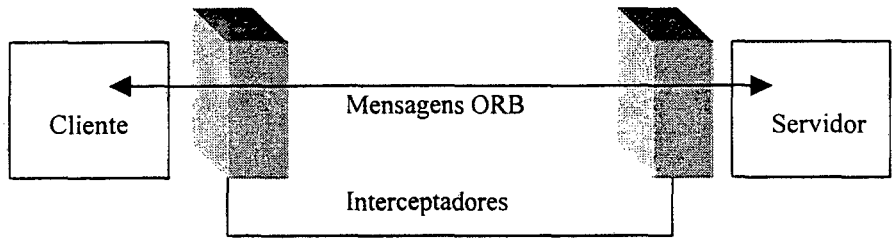

Figura 7.1 - Interceptadores do Visibroker.

7.3 JDK 1.2.2

O JDK (Java Development Kit) é fornecido pela *Sun*. Ele é composto de um conjunto de programas tais como compilador, interpretador, visualizador de *Applet*, gerador de documentação e programa de compactação. Ele é capaz de executar aplicações e *Applets* escritos na linguagem de programação *Java*. No momento, O JDK está disponível para as plataformas: *Microsoft Windows* 95 / NT4.00, *Sun Solaris* 2.4, 2.5, 2.5.1, 2.6 SPARC e *Sun Solaris* 2.5, 2.5.1, 2.6 x86.

A pesar de não possuir um ambiente de desenvolvimento visual, o JDK apresenta menos falha na sua utilização. Os programas que o acompanham são totalmente gratuitos. Existe uma grande variedade de documentações e tutorais o que facilita a sua utilização.

Neste trabalho, foi utilizado o *Symantec Visual Café* para o desenvolvimento das interfaces gráficas.

7.4 Conclusão

Os Interceptadores tiveram um papel bastante importante na implementação deste trabalho pois permitiram ter acesso aos *IIOP Requests* do cliente e invocar de maneira transparente o serviço de autorização da base de segurança.

No próximo capítulo é apresentada a implementação de uma proposta de mecanismo de autenticação e de autorização que, além de proporcionar uma segurança no acesso aos objetos-servidores distribuídos e aos seus usuários, permite ter um total controle sobre o acesso a estes objetos. O que implica uma habilidade para auditar o acesso aos objetos-servidores: quem invocou quais operações e de quais objetos. E, sobretudo apresentar uma interface fácil de operar.

Capítulo 8

Mecanismo de Autenticação e Autorização Proposto

8.1 Introdução

Os protocolos de segurança implementados pelos produtos comerciais são ainda incompletos [23], por exemplo, as implementações do *Security Socket Layer* (SSL) fornecem uma criptografia e uma autenticação somente do servidor mas não do usuário [23]. O mecanismo baseado em chaves privadas do *Kerberos* ainda não é viável quando se trata de um número maior de chaves de usuários [23]. O *Kerberos* é muito usado, porém, apresenta algumas limitações de segurança [35, 36, 37], sendo o problema mais grave a vulnerabilidade do seu servidor. A partir do credencial do administrador (root) um intruso pode obter chaves secretas críticas. Com estas chaves na mão, ele pode revelar toda a informação protegida do servidor *Kerberos*. Além disso o *Kerberos* e outros *Frameworks* de segurança, tais como, POXIX e ISO7498-2 não suportam ambientes distribuídos orientados a objetos [38, 39].

Muitos destes protocolos de segurança apresentam uma taxa de *overhead* (atraso) muito grande, devido a complexidade dos algoritmos de criptografia (cifradores) empregados nos seus mecanismos e, também, pelo fato de serem implementados na camada de rede, como é o caso do SSL [34].

Estes problemas e muitos outros demonstram que as técnicas de segurança usadas hoje estão muito longe de resolver os problemas existentes de segurança. Neste sentido, o objetivo desta proposta é implementar um mecanismo na camada de aplicação que reduza os *overheads* e garanta uma segurança no acesso de objetos-servidores distribuídos

e aos seus usuários. Ter um total controle sobre o acesso aos objetos-servidores. O que implica na habilidade para auditar o acesso aos objetos-servidores e quem invocou quais operações e de quais objetos. E, sobretudo, apresentar uma interface fácil de operar. A Figura 8.0 apresenta o ambiente deste mecanismo.

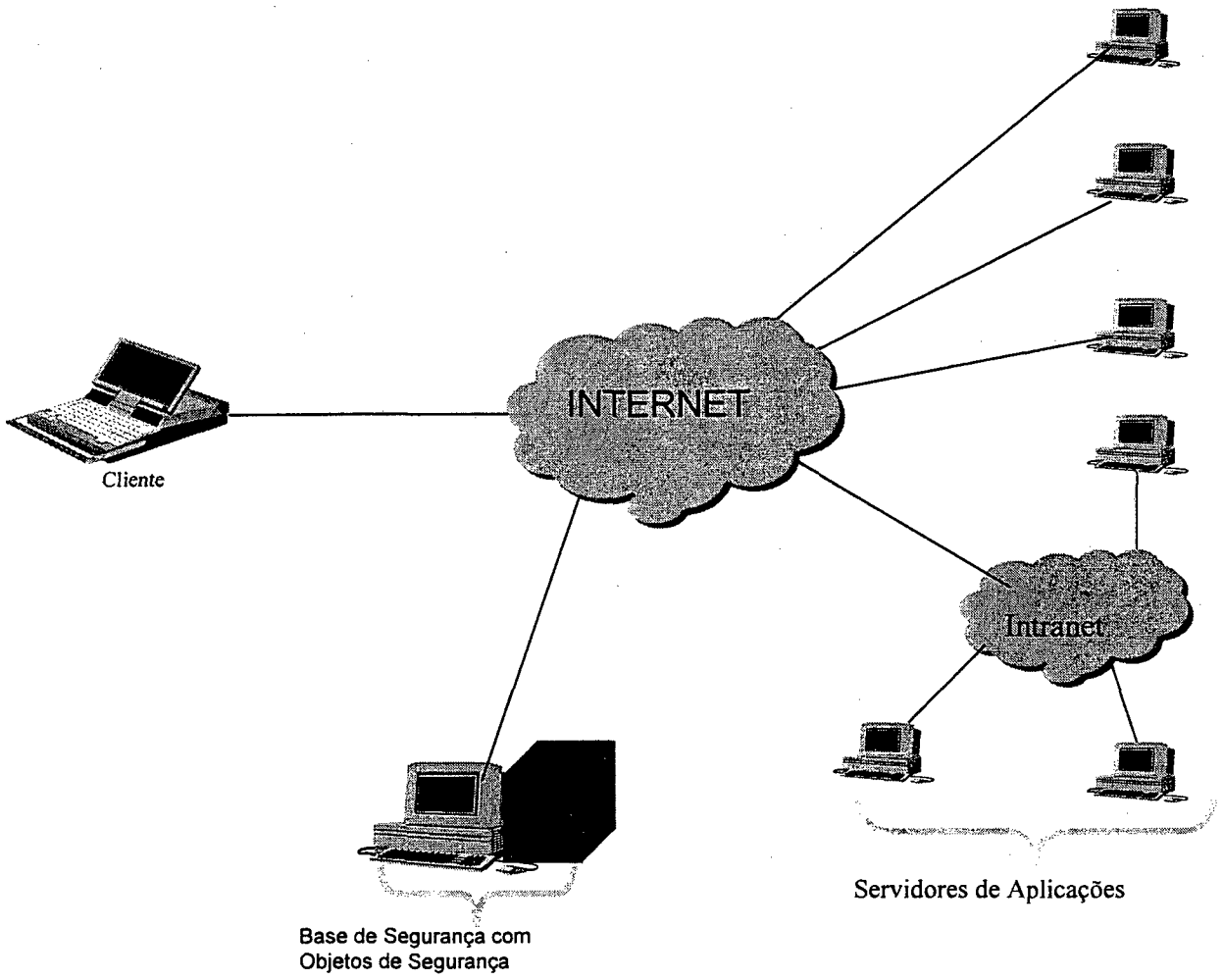

Figura 8.0 Ambiente do Mecanismo de Segurança.

A idéia é que, quando o cliente fornece o seu ID de usuário e a sua senha para entrar numa sessão de aplicação, ele obtenha de maneira transparente um identificador de sessão ou *ticket* gerado pela base de segurança. Este *ticket* é configurado como sendo o nome do *principal*. Quando o cliente faz uma invocação em um objeto, o pseudo-objeto *principal* é

enviado junto com a solicitação através do ORB (Object Request Broker). Quando a solicitação chega no lado do servidor, ela é interceptada por uma autoridade de segurança (interceptador de segurança) que vai solicitar ao objeto de autorização da base de segurança, o controle de acesso verificando se o principal tem direito de invocar o objeto.

8.2 Descrição Funcional do Mecanismo

O mecanismo é composto dos objetos: base de segurança, autenticação e autorização, os quais implementam os serviços de autenticação e de autorização. O cenário deste mecanismo pode ser visto na Figura 8.1.

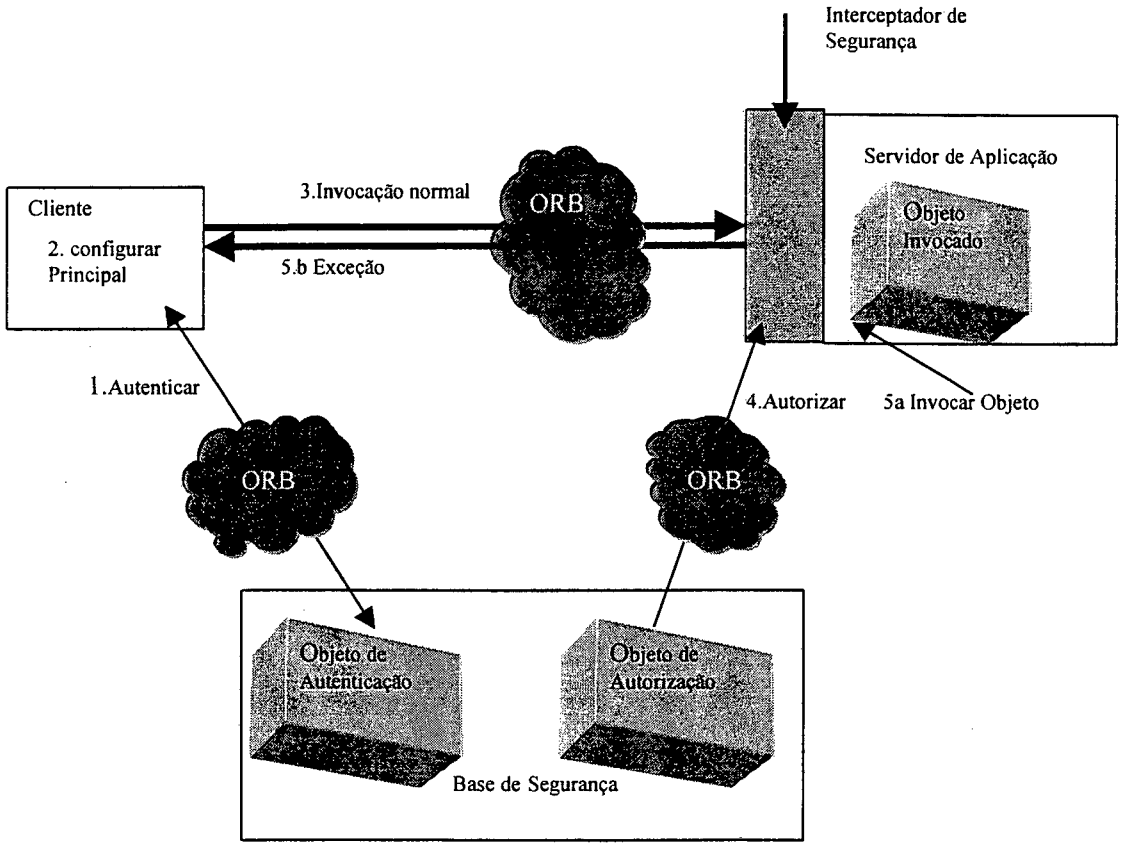

Figura 8.1 - Cenário do Mecanismo de Autenticação e Autorização.

a) Objeto de autenticação

É responsável por efetuar as funções primárias da gerência de segurança dos objetos-servidores. Obter o ID do usuário e verificar se o usuário é realmente quem diz ser. Isso é feito através da sua senha. Se a operação de autenticação for satisfatória é criado um *ticket* configurado como *principal*. A Figura 8.2 descreve o processo de autenticação.

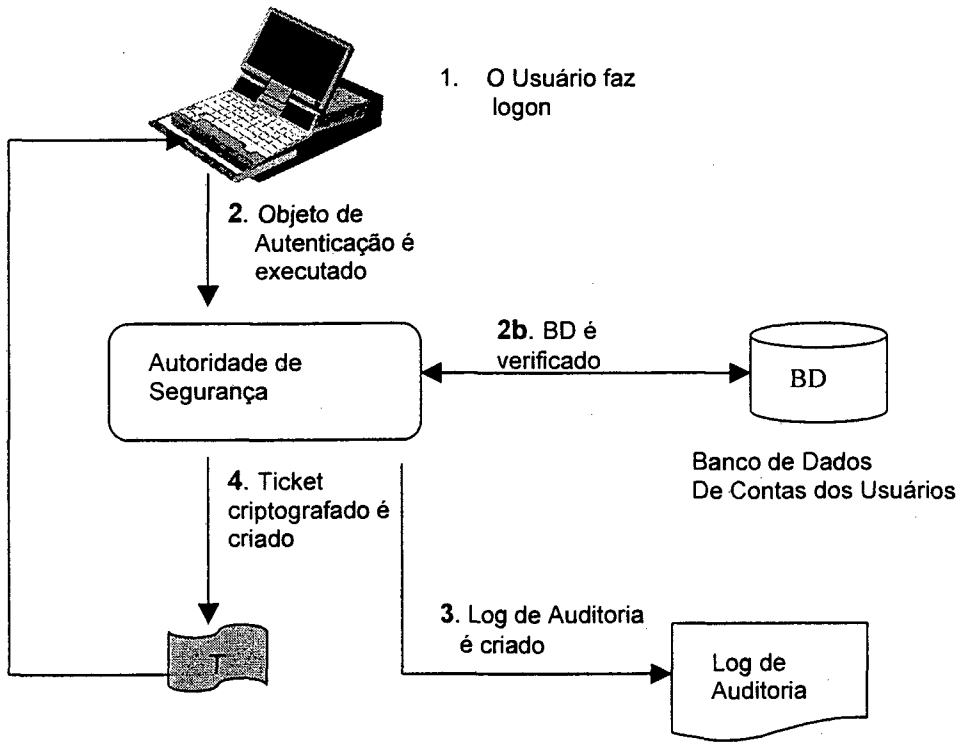

Figura 8.2 Processo de Autenticação

- Processo de autenticação

1. O *principal* fornece seu nome e senha. O nome é usado para a identificação e a senha para a autenticação.
2. O objeto de autenticação é executado para identificar e autenticar o *principal*.
3. Um arquivo *log* é gerado para a realização de auditoria do sistema de segurança.
4. Se a operação de autenticação for satisfatória, o objeto de autenticação gera um *ticket* que contém as informações criptografadas do *principal*. O *ticket* é retornado ao *principal*. Neste ponto, o *principal* é identificado para o sistema de segurança e pode começar a

acessar objetos baseado no controle de acesso discricionário para esses objetos. O ID e a senha são respectivamente verificados. Se existirem, então é criado o *ticket* criptografado. Caso contrário, uma exceção de falha é gerada. O *ticket* criado é então enviado para o *principal* que o configura como sendo seu nome. Veja o Fluxograma do Procedimento de Autenticação na Figura 8.3.

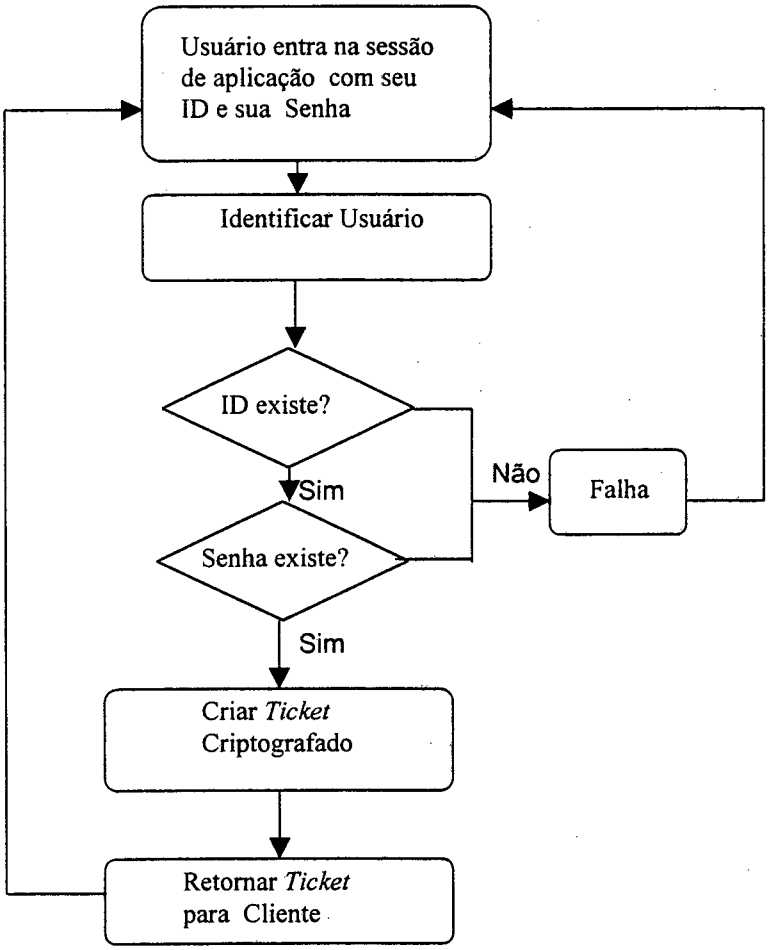

Figura 8.3 - Fluxograma do Procedimento de Autenticação.

b) Objeto de Autorização

Tem como função prover o controle de acesso, a autorização da requisição de acesso ao objeto-servidor. Essa tarefa é realizada de acordo com a política de acesso. Se tal *principal*

estiver cadastrado com direito de invocar operações deste objeto então seu acesso é permitido. A Figura 8.4 apresenta o processo de autorização.

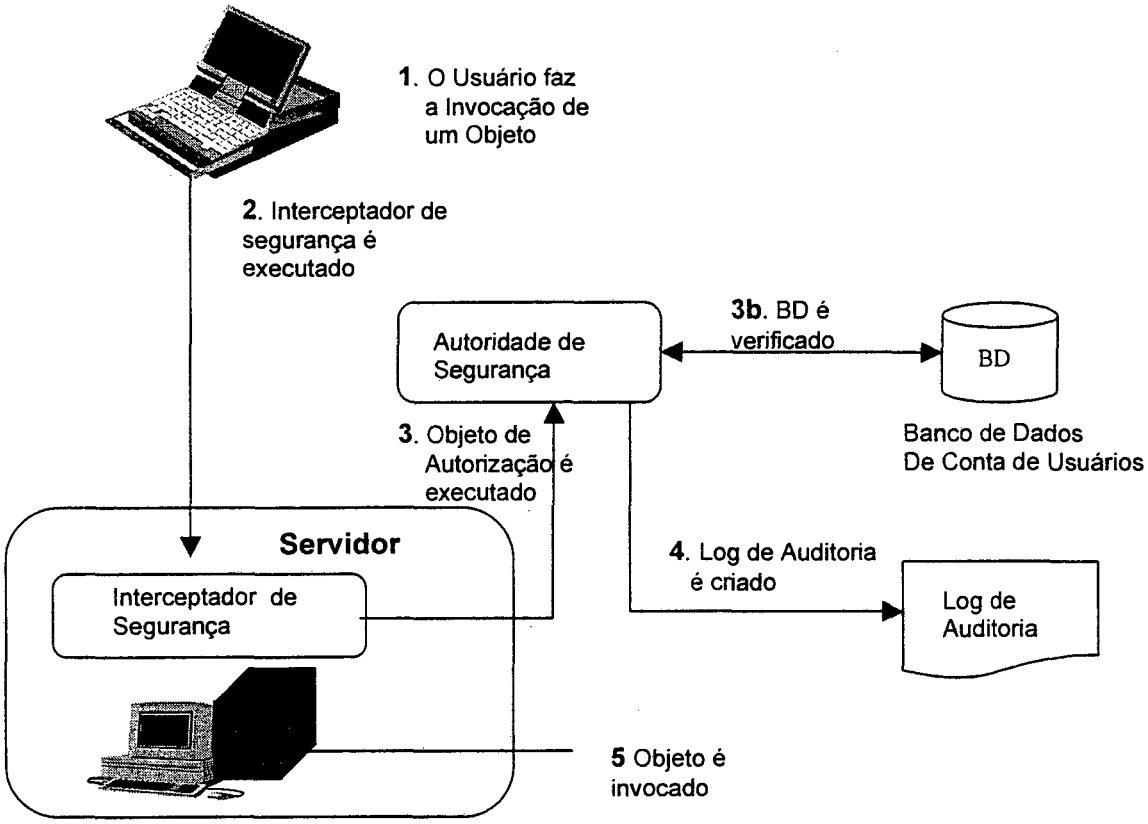

Figura 8.4 - Processo de Autorização.

- Processo de Autorização

O controle de acesso discricionário capacita o principal a ter acesso a um objeto-servidor e, conseqüentemente, invocar suas operações.

1. *Principal* identificado pelo objeto de autenticação invoca um objeto.
2. A invocação é interceptada por um interceptador de segurança instalado junto ao servidor. O interceptador entrega, a invocação do principal à autoridade de segurança para o controle de acesso.
3. O Objeto de Autorização executa o controle de acesso e a autorização da requisição de acesso aos objetos-servidores, isso de acordo com a política de acesso.

4. Um arquivo log também é gerado para a auditoria de sistema.
5. Se o principal estiver cadastrado com o direito de invocar operações desse objeto então seu acesso é permitido. A Figura 8.5 mostra o fluxograma do procedimento de Autorização.

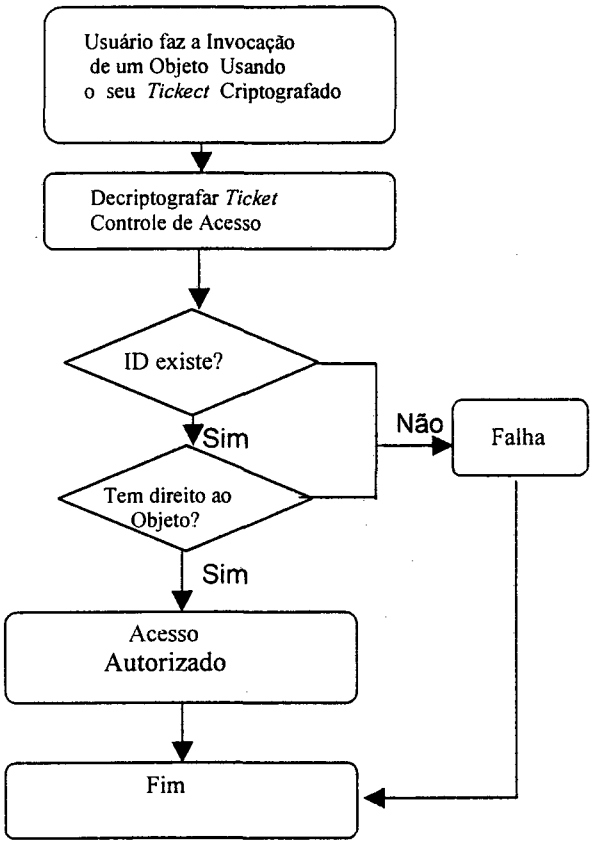

Figura 8.5 - Fluxograma do Procedimento de Autorização.

No procedimento de autorização o *ticket* é decriptografado, o ID do principal e seus direitos para acessar os objetos são respetivamente verificados. Se a operação for satisfatória então o acesso é autorizado, caso contrário é gerada uma exceção de não autorização.

8.2.1 Política de Acesso

A política de acesso implementada nesta abordagem é baseada nos nomes dos objetos-servidores e os atributos dos *principais* tais como a identidade e a sua *capability.* A *capability* identifica os objetos e as operações que o principal tem direito de acessar. A implementação da lista de controle de acesso poderia ser sofisticada com a criação de um banco de dados mas, por razão de simplicidade, optamos por um arquivo texto. O arquivo tem uma linha para cada usuário no seguinte formato.

<Nome do usuário>":"<Senha>":"<Nome do objeto permitido>":" < Nome do objeto permitido > ":"...

Veja um exemplo do arquivo texto no Quadro 1. O asterisco define um direito de acesso a todos os objetos. Note que os atributos são separados por um delimitador ":".

Quadro 1- Lista de Controle de Acesso.

```
Mirela:k4aLYd:saldo extrato
Desire:0fDX&l:*
Carla:hillmann:saldo
```

8.3 Especificação das Interfaces (Objetos)

Na modelagem do mecanismo de segurança são definidos dois objetos distribuídos responsáveis pelos serviços de segurança que foram implementados.

Utiliza-se a Linguagem de Definição de Interface IDL do CORBA especificada pelo OMG para definir as Interfaces do objeto de Autenticação e de Autorização. Na interface de autorização desenvolve-se um método de autenticação, que se baseia no ID do usuário e na sua senha para criar o *ticket* do cliente. Este *ticket* serve como nome do *principal.*

A interface de autorização oferece um método de autorização baseado no *ticket* gerado pelo objeto de autenticação, o objeto que o cliente quer acessar e o nome da operação a ser

invocada. Declara-se também duas exceções *NotAuthorized* e *NotAuthenticated* executadas no caso dos serviços de autenticação e de autorização não ocorrerem corretamente. Veja o Quadro 2. Este quadro ilustra as interfaces IDL.

8.3 Implementação

O Quadro 2 apresentado nesta seção constitui a especificação das interfaces utilizadas no mecanismo de segurança. São definidas três interfaces correspondendo aos objetos utilizados.

Quadro 2 - Especificações IDL dos Objetos da Base de Segurança.

```
module security {
    typedef sequence< octet > Ticket;     // Ticket é uma sequencia de bytes
                                          // ou conjunto nao limitado de bytes
       exception NotAuthenticated{};        // excecao vazia
    exception NotAuthorized{};
    interface Authentication {            // servico de autenticacao
        Ticket authenticate(
                  in string userId,
                  in string passWord )
            raises( NotAuthenticated );
    };
    interface Authorization {             // servico de Autorizacao
        void authorize(
                 in Ticket clientTicket,
                 in Object targetObject,
                 in string operationName )
            raises( NotAuthorized );
    };
    interface BasicSecurity: Authentication, Authorization {}; // herda dos servicos

// de Autenticacao e autorizacao
};
```

8.4.1 Implementação da Interface da Base de Segurança

A base de segurança foi implementa como uma classe Java. Foram declaradas duas tabelas *Hash*: uma, para os usuários e suas senhas, e a outra, para a lista de controle de acesso. O construtor da classe registra o objeto com o *OSAgent,* cria e inicializa as tabelas Hash. Lê o arquivo texto contendo as informações de segurança e as preenche. Para mais detalhes

consulte o código fonte da classe *BaseSecurityImpl* em anexo. Nesta classe foram implementados os dois principais métodos: *authenticate* e *authorize*

- Método *authenticate*

Este método tem como parâmetros o *userID* e o *passWord*. Ele procura o *password* na tabela *Hash, UserTable* se encontrar então cria um *Ticket*. A criação do *Ticket* consiste em criptografar o *userID*. A final o método retorna o valor *Ticket*. Veja o Quadro 3.

Quadro 3 - Método de Autenticação.

```
public byte[] authenticate( String userId, String passWord )
    throws NotAuthenticated {

        if( userTable.containsKey( userId ) &&
          userTable.get( userId ).equals( passWord ) ) {

            // cria o ticket
      return encrypt( userId );
    }
    else {
      throw new NotAuthenticated();
    }
  }
```

- Método *authorize*

O método *authorize* tem três parâmetros: o *Ticket*, o objeto que o cliente quer acessar e o nome da operação a invocar.

Primeiramente, o *Ticket* recebido é descriptografado. Com o *userID* obtido a partir do *Ticket*, a tabela *Hash accesControlelList* é processada para obter o *string* que contém o nome dos objetos que o usuário pode invocar. Se o nome do objeto que o cliente quer invocar estiver dentro deste *string* o método retorna e permite que a invocação prossiga. Isso é também válido se o usuário tem um privilégio global. Caso contrário, a autorização é negada.Veja o Quadro 4 .

Quadro 4 - Método de Autorização.

```
// metodo para autorizar o userID a invocar a operacao
  public void authorize( byte[] ticket, org.omg.CORBA.Object object,
    String operationName )
    throws NotAuthorized {
        String userId = decrypt( ticket );
    System.err.println( "authorize: " + userId );
    String objectId = object._object_name();
    if( objectId == null ) {
      System.err.println( "Nao ha nome de Objeto" );
      return;
    }

    if( accessControlList.containsKey( userId )) {
      String allowedObjects =
        (String) accessControlList.get( userId );
      if( allowedObjects.regionMatches( 0, "*", 0, 1 ) ) {
        return;
      }
      if( allowedObjects.
        regionMatches( 0, objectId, 0, objectId.length() ) ) {
        return;
      }
    }
        throw new NotAuthorized();
  }
```

8.4.2 Interceptador

Os interceptadores do *Visibroker* são estruturas que permitem interceptar, em certos pontos, as solicitações e as respostas do cliente e do servidor. Utilizando estes interceptadores, pode-se ver e modificar a comunicação entre o cliente e o servidor, alterando, assim, o comportamento do ORB. Para mais detalhes consulte o capítulo 16 de [03]. Essas estruturas foram muito importantes no desenvolvimento deste trabalho.

Foi implementado um Interceptador chamado classe *SecurityServerInterceptor.* O construtor dessa classe obtém, a referência da base de serviços de segurança. O método *receive_request()* dessa classe, (veja o quadro 5) permite obter os dados necessários para o controle de acesso, ou seja, a autorização.

Quadro 5 - Invocação do Serviço de Autorização.

```
// obter a partir do IIOP request o principal, objeto de destino e a operação
  // invocada e invocar o serviço de autorização
```

```
public InputStream receive_request(RequestHeader hdr,
            org.omg.CORBA.ObjectHolder target,
            InputStream buf, Closure closure)
{
   try {
      basicSecurity.authorize(
         hdr.requesting_principal,
         target.value,
         hdr.operation );
      return null;
   }
   catch( NotAuthorized na ) {
         System.err.println("not authorized");
         throw new NO_PERMISSION();
   }
}
```

São extraídos o *principal* e o nome da operação invocada a partir do *RequestHeader* e a referência do objeto-destino a partir do *HolderObject*. Esses valores são passados como argumentos na invocação da operação de autorização na base de serviços de segurança.

O *RequestHeader* e o *HolderObject* são classes padrão do CORBA e do interceptador do *Visibroker*.

8.5 Testes

A aplicação-cliente foi implementada com um *Applet Java.* (Veja a Figura 8.6). Em conseqüência deste fato, ela possui a característica de ser independente de plataforma e de poder ser utilizada a partir de qualquer computador pertencendo a Internet, sem a necessidade da instalação previa de algum *software.*

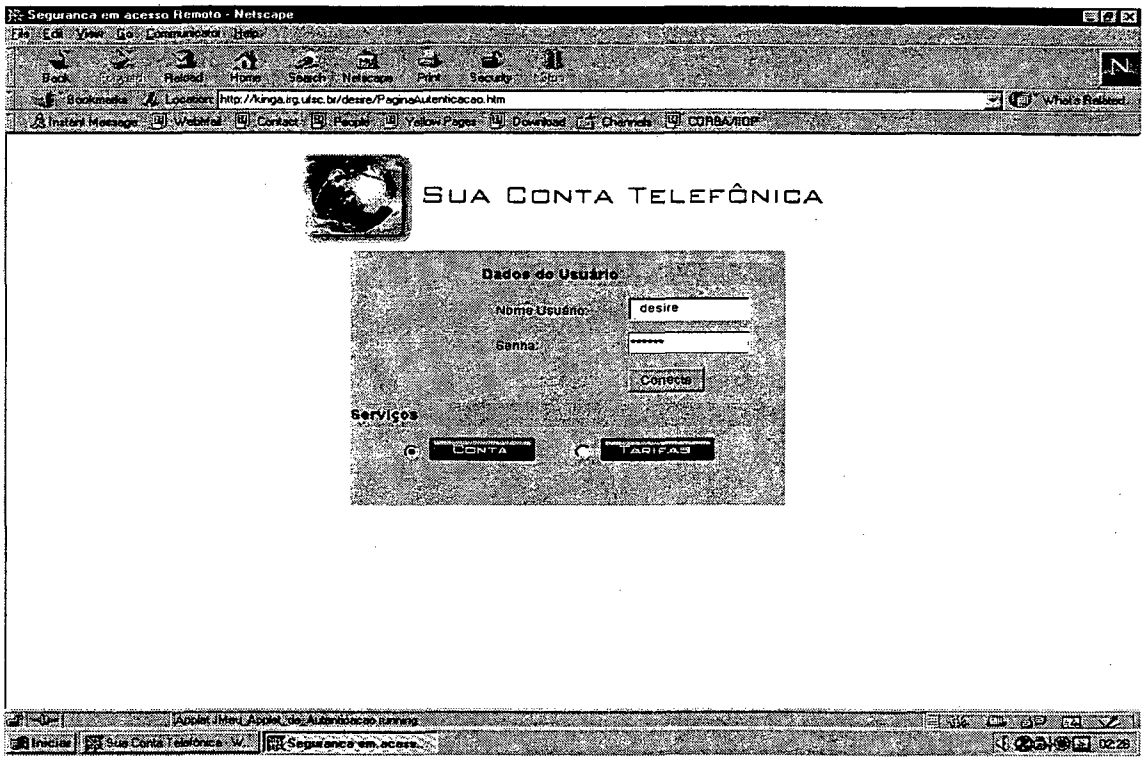

Figura 8.6 - Interface do Cliente.

A Figura 8.6 também apresenta o instante em que o usuário fornece seus dados e solicita um determinado serviço do objeto-servidor. A conexão dispara automaticamente os serviços de segurança que são executados no servidor da base de segurança.

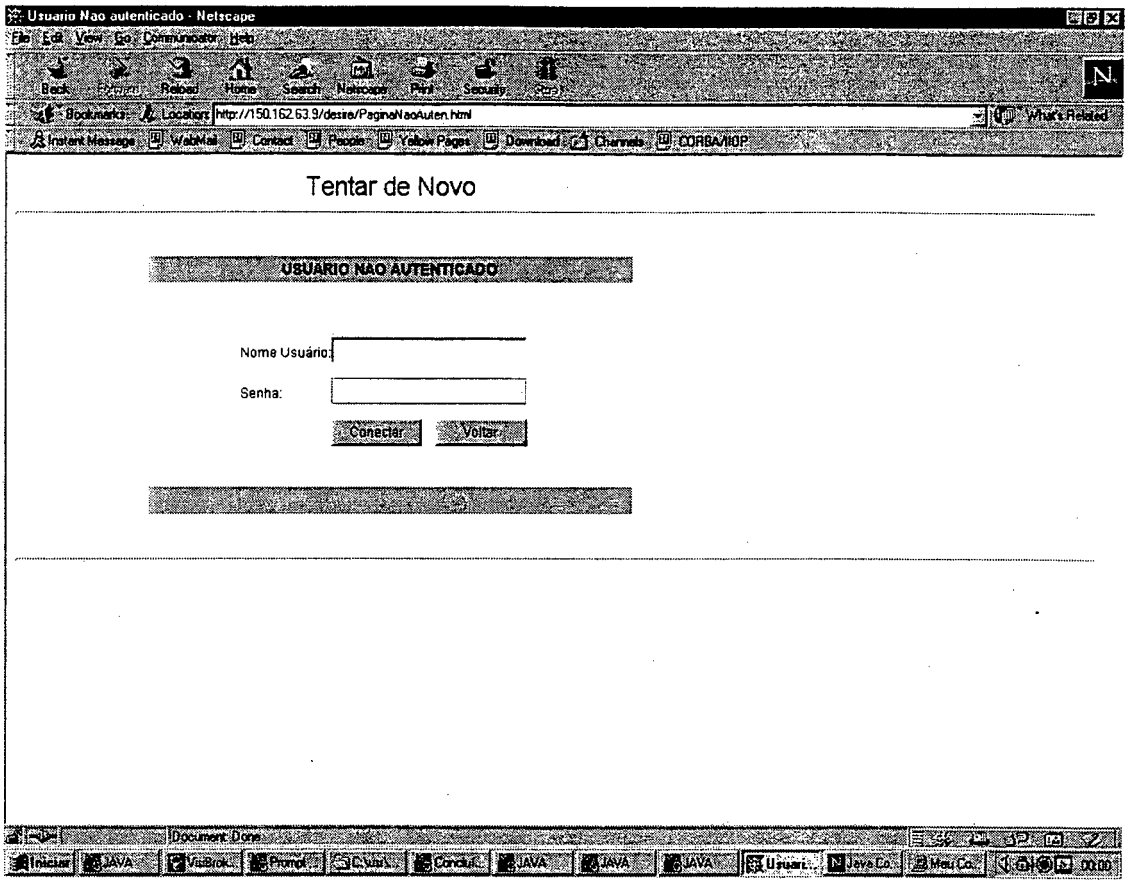

Figura 8.7 - Interface do Cliente com a Exceção de Usuário não Autenticado.

As figuras: Figura 8.7 e Figura 8.8 são páginas WWW com *Applet Java.* Elas apresentam as situações em que o usuário não é autenticado ou autorizado a invocar um serviço do objeto-servidor.

Com a utilização do CORBA, a implementação do objeto de autenticação e de autorização é executada no servidor onde eles estão instalados. O *applet* do cliente necessita apenas conter as definições das interfaces destes objetos. Isso simplifica bastante a implementação da aplicação-cliente.

O serviço é executado no servidor e os resultados trazidos no browser do cliente. Isso traz inúmeras vantagens. Uma delas é que, as operações realizadas pelo objeto, no servidor, não são sujeitas às limitações de segurança impostas à *applet*, sendo executado em *browsers Web*. Uma outra vantagem é que a máquina do cliente não requer um grande recurso computacional pois o esforço computacional, é feito pelo servidor. A Figura 8.9 apresenta um extrato de conta telefônico. O serviço de extrato de conta telefônico é implementado por um objeto do protótipo SETWeb desenvolvido pela equipe [28]. Neste teste, um dos objetivos é garantir um controle de acesso ao objeto SETWeb. Veja a Figura 8.10.

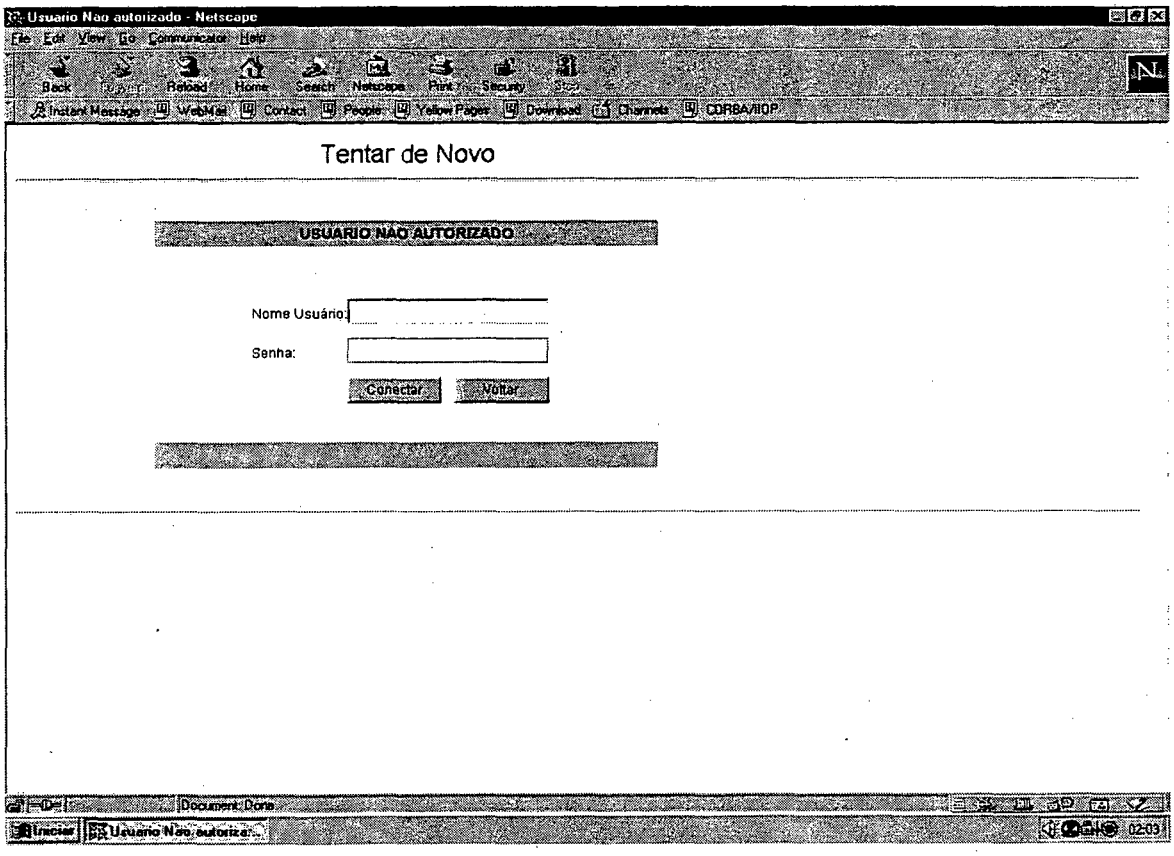

Figura 8.9 – Interface do Cliente com a Exceção de Usuário não Autorizado.

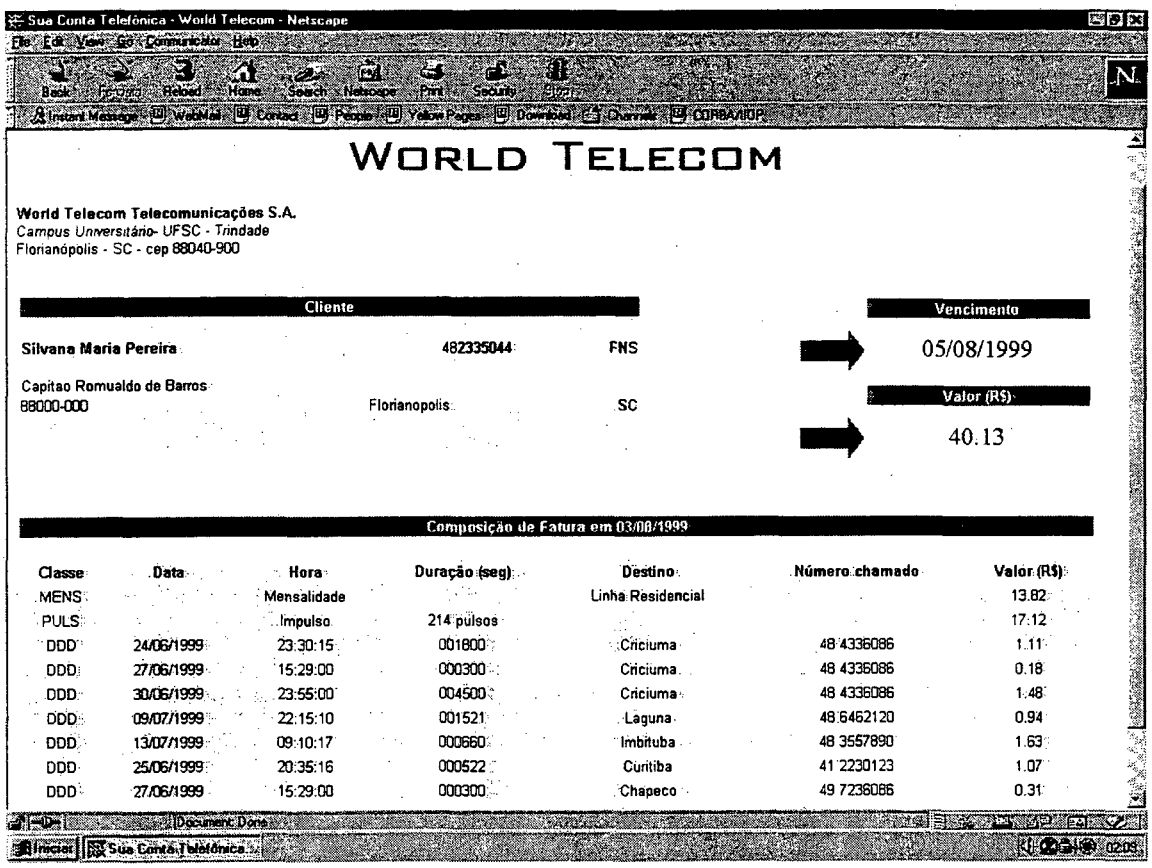

Figura 8.10 - Extrato de Conta Telefônica.

8.6 Conclusão

Neste capítulo foi apresentada a descrição funcional do mecanismo, relatada a política de acesso e descrita a implementação dos objetos de autenticação, de autorização e o interceptador. O próximo capítulo apresenta as conclusões e as perspectivas futuras.

Capítulo 9

Conclusões e Perspetivas Futuras

O objetivo principal definido para este trabalho é o de explorar a gerência de segurança no âmbito de gerenciamento de sistemas distribuídos. Neste contexto um mecanismo de autenticação e autorização de acesso a objetos distribuídos foi implementado. A implementação considera sistema distribuído baseado em objetos distribuídos segundo OMG-CORBA e as tecnologias da *Internet* como *Java, Applet Java, Web* e *Browser*.

Neste sentido iniciou-se uma série de estudos baseados na literatura e em material obtido junto com a *Internet* objetivando o domínio destas tecnologias. Na primeira etapa foi feito um estudo aprofundado do modelo de referência de segurança do CORBA definido pelo OMG. Este modelo serviu de inspiração para o mecanismo de segurança proposto neste trabalho. Para um melhor entendimento dos conceitos e fundamentos desta proposta, achou-se também necessário apresentar e discutir neste trabalho este modelo de referência do OMG.

Após esses estudos chegou-se nas seguintes conclusões:

- O OMG buscou incluir no seu modelo de segurança todas as possíveis funções que uma aplicação pode requerer, porém, este modelo é muito complexo e genérico de mais. Algumas funcionalidades não são suportadas pelas tecnologias que implementam a segurança;

- As atuais revoluções na *Internet* e na computação de objetos distribuídos trilham o mesmo caminho. A *Internet,* vista como um *framework* de comunicações, fornece uma plataforma ideal para aplicação de objetos distribuídos. Ao mesmo tempo, a tecnologia de objetos distribuídos está melhorando a qualidade das aplicações baseadas em *Web*, dando mais importância à *Internet* e à *Intranet* nas empresas de Tecnologia de Informações (TI). Esse relacionamento simbiótico entre as duas tecnologias cria um novo modo de conceituar, projetar, desenvolver, executar e manter as aplicações de negócios. Isso faz surgir um novo paradigma de aplicação, construída a partir de códigos fontes encapsulados na forma de objeto executado em sistema protegido, e interagindo uma com a outra através de um ORB. Este novo paradigma de aplicação tem muitos benefícios a oferecer às empresas de Tecnologia de Informações: como, por exemplo, flexibilidade para integrar aplicações de diferentes fabricantes, administração e gerenciamento centralizados dos objetos aplicativos, drástica redução no custo da configuração e manutenção dos sistemas clientes, dentre outros. As empresas que empregam ou se adaptam a estes novos conceitos, sem dúvida terão uma grande vantagem sobre seus competidores.

Com a base teórica adquirida nesses estudos (estudos baseados na literatura e em material obtido junto com a *Internet*) e a escolha da arquitetura CORBA para o desenvolvimento do mecanismo proposto, iniciou-se o estudo de uma ferramenta que possuisse implementação de um ORB e permitisse a integração do CORBA com Java. Com o fruto da pesquisa realizada foi selecionado O *Visibroker 3.4.* Com a utilização desta ferramenta tornou-se viável a definição das interfaces para as funções de segurança. Foram, então, definas as interfaces de autenticação de autorização e da base de segurança. Cada uma delas fornece uma operação associada a uma função de segurança. A interface da base de segurança herda as interfaces de autenticação e autorização. A idéia de definir esta terceira interface está relacionada à capacidade de encapsular os objetos de segurança num único objeto, chamado base de segurança, que atua como uma autoridade de segurança.

Após definir claramente as interfaces e as políticas de segurança associadas à base de segurança, as interfaces foram implementadas utilizando o JDK1.2.2

Um dos objetivos no desenvolvimento deste trabalho é associar a tecnologia *Web* com a arquitetura CORBA. Neste sentido, o aplicativo-cliente foi desenvolvido na forma de um *Applet*. Um cliente pode invocar um objeto-servidor, a partir de qualquer computador pertencente a rede *Internet*, onde a segurança do objeto-servidor é feita de maneira transparente pela autoridade da base de segurança.

9.1 Resultados Esperados e Obtidos

A segurança é um assunto importante para as aplicações distribuídas (objetos distribuídos), este assunto se torna ainda mais importante por ser um requisito básico quando estes objetos são executados através da rede *Internet*.

Neste trabalho, era esperado prover uma segurança aos objetos-servidores distribuídos e a seus usuários. Os serviços de autenticação e de autorização implementados asseguram que o responsável (usuário) pela invocação de um objeto é realmente quem ele diz ser, permite o acesso de um determinado usuário a objetos previamente destinados ao mesmo. Os testes realizados na aplicação do mecanismo no protótipo SETWeb comprovam estes resultados. O mecanismo proposto é aberto. Outros serviços de segurança podem ser acrescentados no futuro. O objetivo principal deste trabalho foi alcançado com sucesso.

9.2 Dificuldades Encontradas

A dificuldade inicialmente encontrada para o desenvolvimento deste trabalho foi a grande quantidade de conhecimentos teóricos exigidos para sua implementação. Poucos trabalhos

encontrados sobre a segurança de computador e de rede foi uma preocupação inicial do projeto.

Várias tecnologias, como por exemplo, a *Internet*, a orientação a objetos, o paradigma cliente/servidor, a arquitetura CORBA e os ORBs foram estudadas. Este estudo envolve a necessidade de uma vasta pesquisa bibliográfica que, em alguns dos casos, não pode ser realizada com a profundidade desejada.

Na fase da implementação outros problemas foram encontrados, tais como a configuração das variáveis de ambiente para o uso adequado do JDK, a linguagem de programação *Java,* pois foi iniciado este trabalho sem nenhum conhecimento prévio da linguagem *Java.* Outros problemas enfrentados foram: a integração do *Java* com CORBA e a incompatibilidade entre as versões de *softwares.* Por exemplo o *Visibroker 3.4* não é compatível com as versões inferiores do JDK 1.2, o *Gatekeeper* na sua versão 3.3 foi implementado para o JDK1.1.8. Essas dificuldades e muitas outras foram superadas através das listas de discussões, pois a documentação destas ferramentas de avaliação disponibilizadas muitas vezes eram incompletas.

9.3 Trabalhos Futuros

A arquitetura do mecanismo de segurança proposto é bastante aberta, portanto, outros serviços de segurança tais como integridade, e não-repudio podem ser desenvolvidos e acrescentados futuramente ao modelo.

Avaliar o desempenho e a performance do modelo na sua integração final com o SETWeb e o SIPI sistemas que estão sendo desenvolvidos pelo Grupo de Rede e Gerência (GRG) do Laboratório de Redes e Gerência (LRG) da Universidade Federal de Santa Catarina (UFSC).

As pesquisas da utilização conjunta do CORBA, Java e WWW para gerência de redes, estão apenas começando. As discussões apresentadas e a proposta do modelo de gerência de segurança apresentado neste trabalho contribuem para a realização de gerência de redes distribuídas.

10. REFERÊNCIAS BIBLIOGRÁFICAS

[01] OMG Document, number 98-12-09. ***Security Service Specification,*** in CORBAservices: Common Object Services Specification. 1998. (http://www.omg.org/).

[02] OMG Document, number 98-12-09. ***CORBA/IIOP 2.2 Specification,*** in CORBAIIOP Specification, December 1998. (http://www.omg.org/corba/corbiip.htm.).

[03] Inprise Corporation. ***Visibroker™ for Java ™*** , Programmer's Guide version 3.3, 100 Enterprise Way Scotts Valley, CA 95066-3249, June 1998.

[04] Jean, Marc Geig; Christophe, Gransart; Phillipe, M.. ***CORBA: des Concepts à la Pratique,*** Collection Inter Editions. Edition Maison. Paris. France.1997.

[05] Kenneth, P. ***Building Secure and Reliable Network Application***, ACM Computer Surveys, vol. 25. No. 2 , June 1998.

[06] Orfali, R.; Harkey, D.. ***Client/Server Programming with Java and CORBA,*** New York. Ed. John & Wiley. 1997.

[07] Haggerty P.; Seetharaman, K.. ***The Benefits of CORBA-Based Network Management,*** IEEE Communications of the ACM. vol. 41. 1998. pp. 73-79.

[08] Vogel, Andreas Keith Duddy. ***Java ™ Programming with CORBA,*** New York, Ed John & Wiley. 1997.

[09] Cameron, D.. ***Security Issues for the Internet and the World Wide Web***. Computer Technology Research Corp. South Carolina, USA. 1996.

[10] Westphall, Carla M. Monografia do exame de Qualificação. ***Esquemas de Autorização para a Programação Distribuída Combinando os Modelos de Segurança Java/CORBA/Web,*** UFSC-LCM Florianópolis, julho de 1998. Certificado de Registro n. 165.663, Livro 277, Folha 4 da Fundação Biblioteca Nacional, Ministério da Cultura, Escritório de Direitos Autorais, Rio de Janeiro.

[11] Patrick, W; John, T.; Machenry. ***Network Security it's Time to Take Seriouly,*** IEEE Communications Magazine, February 1997.

[12] Orfali, R.; Harkey D; Edwards J.. ***The Essencial Distributed Object-SurvivalGuide,*** John Wiley & Sons, Inc, 1996.

[13] Orfali, R.; Harkey D;Edwards J. ***Client/Server Programming,*** *John* Wiley & Sons, Inc, 1996.

[14] Curtis, D.. ***Java, RMI and CORBA,*** OMG White Paper. (http://www.omg.org/paper).

[16] Andre, Mello Barotto. Dissertação de Mestrado. ***Realização da Gerência Distribuída de Redes Utilizando SNMP, Java, WWW e CORBA,*** CPGCC/UFSC Florianópolis, abril 1998.

[17] Alexandre, Veloso de M.. Dissertação de Mestrado ***Gerência de Segurança em Aplicações de Banco de Dados na Web***. CPGCC/UFSC. Março, 1999.

[18] Wallach, D.; Balfanz, D.; Dean D. et al.. ***Extensible Security Architectures for Java***, Proceedings of the 16th Symposium on Operating Systems Principles. Saint-Malo, 1997. pp 116-128.

[19] Gong, L.; Schemers, R.. ***Signing, Sealing, and Guarding Java™ Objects,*** in Mobile Agents and Security. Springer-Verlag. Editor G. Vigna. LNCS 1419. Pp. 206-216. Berlim. 1998.

[20] Sun MicroSystems. ***JDBC™ Guide: Getting Started***, Mountain View, EUA, 1997, (http://java.sun.com/products/jdbc).

[21] Dogac, A. et al.. ***METU Interoperable Database System***, In METU wite paper. December, 1998. (http://www.srdc.metu.edu.tr).

[22] Tari, Z.; Cheng, W.; Yetongon, K. et al.. ***Towards Cooperative Databases: The DOK Aproach***, Proceedings of the International Conference on Parallel and Distributed Computing Systems. Dijon. França.1996. pp 595-600.

[23] Guttman, B.; Bagwill, R.. ***Internet Security Policy: A Technical Guide,*** NIST special publication number 800-XX.Gaithersburg. July, 1997 (http://csrc.nist.gov/isptg/html/).

[24] James Martin ; James J. Odell. ***Análise e Projeto Orientados a Objeto***, Editora Makron Books, São Paulo, 1995.

[25] David White. ***Distributed Systems Security,*** From DBMS, Intelligente Enterprise, November, 1998.

[26] Peter, Coad; Edward, Youdon. ***Projeto Baseado em Objeto,*** 4ª edição, Editora Campus, Série Yourdon Press, Rio de Janeiro, 1993.

[27] Wiliam, Stallings. ***Cryptography and Networks Security Principales and Pratices,*** IEEE Communication magazine, july 2000.

[28] Fernanda, Pereira de Souza; Carla Cristina Leite. ***Sistema de Extrato Telefônico via Web em Java com Suporte a CORBA,*** Monografia de conclusão de curso Novembro 1999.

[29] José, Eduado de Lucca. Dissertação de Mestrado. ***Arquitetura de Segurança para Rede Aplicada a Sistemas de Gerência,*** CPGCC/UFSC BU/DPT 0-239-131-6.

[30] Quoin. ***Distribuited Computing Overview,*** Cambridge, Massachussetts (http://www.quoin.com).

[31] Borland Inprise. ***Distributed Object Computing in the Internet Age,*** Inprise Corporation. 100 Enterprise Way Scotts Valey, CA. 1998 (http://www.borland.com/visibroker/paper/distributed/wp.html).

[32] Inprise Corporation, 95066-3249. ***Visibroker for Java Gatekeeper Guide Version 3.3.*** 100 Enterprise Way Scotts Valey, CA. 1998

[33] Lappinen, Mika;Pulkkinen, pekka; Rautiainen,Aapo. ***Java and CORBA Based Network Management.*** IEEE Computer, June1997. (http://dlib.computer.org/).

[34] Netscape white paper. ***The SSL Protocol, Cipher Used with SSL and the SSL Handshake,*** http://developer.netscape.com/doc/manuals/security/sslin/contents.htm.

[35] Bellovin, S.; Merritt. ***Limitation of Kerberos Authentication System*** in Processings of Winter 1991 Usenix Conference, January 1991. ftp://research.att.com/dist/internet_security/kerblimt.usenix.ps.

[36] Naomaru, Itoi; Peter Honeyman. ***Smartcard Integration with Kerberos V5,*** in Processings of USENIX Workshop on Smartcard Technology, Chicago, may 1999.

[37] Peter A.; Loscocco; Stephen D.; Smalley,et al... ***The Inevitable of Failure: The Flawed Assumption of Security,*** in Modern Computing Environments, in 21st National Information Systems Security Conference, Crystal City, Virginia, October 1998. National Security Agency , NISSC. http://www.jya.com/paperF1.htm.

[38] Brennan, R.; Jennings, B.; McArdle, C, et al.. ***Teletec Irland, Evolutionary Trends in Intelligent Networks,*** IEEE Communications Magazine, June 2000.

[39] Mampaey, M.; Couturier A.; Alcatel. ***Using TINA Concepts for Evolution***, IEEE Communications Magazine, June 2000.

11. Anexos

Anexo A : Código Fonte da Implementação da Base de Segurança

```
/*****************************************************************************
  Pacote          : securityImpl
  Classe          : BasicSecurityImpl.java
  Objetivo        : Implementar a interface da base de segurança
  Implementaçao :
                   Declarar duas tabelas Hash uma contendo os dados do usuário  e a
                   outra para a lista de controle de acesso.
                   Ler um arquivo texto contendo as informações de segurança e
                   preencher as duas tabelas Hash com as informações lidas.
  authenticate():procurar  o password do userID na tabela Hash de usuários
                   se encontrar então criar o Ticket e retornar o Ticket para
                   o cliente.
  encrypt()       criar o Ticket criptografando o userID
                   e retornar userID na forma de  uma sequência bytes cifrados.
  decrypt()       decriptografa o Ticket transformando o byte em string
  authorize()     usar tabela Hash  de controle de acesso para obter os objetos
                   que UserID pode ter acesso. Se o nome do objeto invocado pelo
                   userID estiver dentro da lista desses objetos a invocação é
                   autorizada.

*****************************************************************************
*/
package securityImpl;

import java.util.*;
import java.io.*;
import org.omg.CORBA.*;
import seguranca.*;

public class BasicSecurityImpl extends _sk_BasicSecurity {

    private Hashtable userTable;
    private Hashtable accessControlList;
    private String star;

    BasicSecurityImpl( String name, String fileName ) {

        super( name );
      // criar a tabela hash de usuarios
       userTable = new Hashtable( 50 );
      // criar a tabela hash da lista de controle de acesso
        accessControlList = new Hashtable( 100 );
        star = new String("*");

        try {
            RandomAccessFile file;
            String line;
            String user, passwd, acDesc;
            int userIndex, passwdIndex, acDescIndex;

            file = new RandomAccessFile( fileName, "r" );
```

```
            while( ( line = file.readLine() ) != null ) {

                userIndex = line.indexOf(':', 0);
                if( userIndex > -1 ) {
                    user = line.substring( 0, userIndex );
                    passwdIndex = line.indexOf(':', userIndex+1);
                    if( passwdIndex > -1 ) {
                        passwd = line.substring( userIndex+1, passwdIndex );
                        userTable.put( user, passwd );

                        acDesc = line.substring( passwdIndex+1 );
                        if( acDesc !=null ) {
                            accessControlList.put( user, acDesc );
                        }
                        else {
                            System.err.println( "usuário" + user +
                                " não tem acesso" );
                        }
                    }
                    else {
                        System.err.println( "usuário " + user +
                            " não tem senha - erro no formato do arquivo?" );
                    }
                }
                else {
                    System.err.println( "linha " + line +
                        " não tem usuário - erro no formato do arquivo?" );
                }

            }
        }
        catch ( IOException ioex ) {
            System.err.println( "arquivo " + fileName + " não encontrado" );
            System.exit( 1 );
        }
    }
//  metodo para criptografar o userID

    public byte[] encrypt( String userId )  {

        return userId.getBytes();
    }
//  metodo para decriptografar o ticket
    public String decrypt( byte[] ticket )  {

        return new String( ticket );
    }
// metodo para autenticar userID en Ticket
    public byte[] authenticate( String userId, String passWord )
        throws NatAuthenticated {

      if( userTable.containsKey( userId ) &&
          userTable.get( userId ).equals( passWord ) ) {

          // create a simple ticket
            return encrypt( userId );
        }
        else {
            throw new NotAuthenticated();
```

```
        }
    }
// metodo para autorizar o userID a invocar a operacao
    public void authorize( byte[] ticket, org.omg.CORBA.Object object,
        String operationName )
        throws NotAuthorized {

        String userId = decrypt( ticket );
        System.err.println( "Tentou uma Autorizacao: " + userId );
        String objectId = object._object_name();

        if( objectId == null ) {
            System.err.println( "Tentou a uma Autorizacao: non-name object" );
            return;
        }

        if( accessControlList.containsKey( userId )) {
            String allowedObjects =
                (String) accessControlList.get( userId );
            if( allowedObjects.regionMatches( 0, "*", 0, 1 ) ) {
                return;
            }
            if( allowedObjects.
                regionMatches( 0, objectId, 0, objectId.length() ) ) {
                return;
            }
        }
       throw new NotAuthorized();
    }
}
//Fim ********************************************************
```

Anexo B : Código Fonte da Implementação do Servidor

```
/* ***************************************************************************
  Pacote    : securityImpl
  Classe    : server.java
  Objetivo : Inicializar a ORB e Criar o objeto da Base de
             Segurança BasicSecurityImpl e aguardar
             solicitações

***************************************************************************
*/
package securityImpl;

import org.omg.CORBA.*;

public class Server {

    public static void main(String[] args) {

        if( args.length != 2 ) {
            System.out.println(
            "Eh necessario digitar dois argumentos " +
            "<name>, <Arquivo da lista de controle");
            System.exit( 1 );
        }

        try {
            //init ORB
            ORB orb = ORB.init( args, null );
            BOA boa = orb.BOA_init();

            // cria o objeto BasicSecurity que o objeto da base
            // de seguranca
            BasicSecurityImpl basicSecurityImpl =
                new BasicSecurityImpl( args[0], args[1] );

            // Exporta a referencia do objeto
            boa.obj_is_ready( basicSecurityImpl );

            // inprime a referencia na forma de um string
            //System.out.println( orb.object_to_string( basicSecurityImpl ) );

            // aguarda requisicoes
            boa.impl_is_ready();
        }
        catch(SystemException e) {
            System.err.println(e);
        }
    }
}
// Fim ************************************************************
```

Anexo C : Código Fonte da Implementação do Interceptador no Lado do Servidor

```
/*
*****************************************************************************
*
  Pacote:     securityInterceptor
  Classe:     SecurityServerInterceptor.java
  Objetivo:   Interceptar o IIOP request e extrair os dados contidos nele
  Implementação:
              Obter a referencia do objecto  base de segurança através do método
              bind()
              com o método receive_request obter a partir do IIOP request os dados
              necessários para o serviço de autorização. (principal, operação
              invocada, referência do objeto destino)
              O principal e o nome da operação invocada  são obtidos a partir do
              requestHeader e a referência do objeto destino a  partir  do
holderObjet
*****************************************************************************
****
*/


package securityInterceptor;

import com.visigenic.vbroker.interceptor.*;
import com.visigenic.vbroker.IOP.*;
import com.visigenic.vbroker.GIOP.*;
import org.omg.CORBA.*;
import org.omg.CORBA.portable.*;
import security.*;
//import java.util.*;
//import java.io.*;
public class SecurityServerInterceptor implements ServerInterceptor {

    private BasicSecurity basicSecurity;
    private String literal;
    public SecurityServerInterceptor() {
        // inicializar a ORB
        ORB orb = ORB.init();
        try {
            // obter a referncia do objeto base de seguranca
            basicSecurity = BasicSecurityHelper.bind( orb );
        }
        catch(Exception e) {
            System.err.println(
                " referencia da base de seguranca nao obtida");
            System.err.println(
                "Abortando a execucao ... ");
            System.exit(-1);
        }
    }
    // metodos  invocados para localizar um objeto
    public IOR locate(int req_id,
           byte[] object_key,
           Closure closure) {
        return null;
    }
```

```
    public void locate_succeeded(int req_id,
           Closure closure) {}

    public void locate_forwarded(int req_id,
           IORHolder forward_ior,
           Closure closure) {}

    public IOR locate_failed(int req_id,
           byte[] object_key,
           Closure closure) {
        return null;
    }
    // obter a patir do IIOP request o principal, objeto de destino e a operacao
    // invocada
    public InputStream receive_request(RequestHeader hdr,
                       org.omg.CORBA.ObjectHolder target,
                       InputStream buf, Closure closure)
    {
        try {
            basicSecurity.authorize(
                hdr.requesting_principal,
                target.value,
                hdr.operation );
            return null;
        }
        catch( NotAuthorized na ) {
                /* ***Pode-se criar um arquivo
                      log dos dados do principais
                      nao autorizados tentando acessar
                      o objeto servidor    *************
                */
                //literal = String( hdr.requesting_principal );
                System.err.println(" Nao autorizado");
                throw new NO_PERMISSION();
        }
    }
  /*
*****************************************************************************
*
                Métodos invocados durante um IIOP response
   prepare_reply     : disparado quando uma resposta é preparada
   send_reply        : disparada quando uma resposta é enviada
   send_reply_failed : disparada quando a resposta é enviada
                       sem  sucesso
   send_reply_request: disparada quando a resposta é enviada
                       com sucesso

*****************************************************************************
*
   */
  public void prepare_reply(RequestHeader hdr,
                ReplyHeaderHolder reply,
                org.omg.CORBA.Object target,
                Closure closure)
    {
    }


    public org.omg.CORBA.portable.OutputStream send_reply(RequestHeader reqHdr,
```

```
                ReplyHeader hdr,
                org.omg.CORBA.Object target,
                OutputStream buf,
                org.omg.CORBA.Environment env,
                Closure closure)
    {
        return null;
    }

    public void send_reply_failed(RequestHeader reqHdr,
                ReplyHeader replyHdr,
                org.omg.CORBA.Object target,
                org.omg.CORBA.Environment env,
                Closure closure) {}

    public void request_completed(RequestHeader reqHdr,
                org.omg.CORBA.Object
                target, Closure closure) {}

    // invocado quando a conexao e terminada
    public void shutdown(
    com.visigenic.vbroker.interceptor.ServerInterceptorPackage.ShutdownReason
    reason) {}
    // invocado quando ocorre uma falha na conexao
    public void exception_occurred(RequestHeader reqHdr,
                org.omg.CORBA.Environment env,
                Closure closure) {}
}

// Fim  ************************************************************
```

Anexo D : Códigos fonte da Implementação que cria a *Fábrica* de Interceptadores

```
/**************************************************************************
 Pacote: securityInterceptor
 Nome   : SecurityServerInterceptorFactory.java
 Objetivo: criar uma instancia do ServerInterceptor quando ela nao existe
 **************************************************************************
*/
 package securityInterceptor;

import com.visigenic.vbroker.interceptor.*;

public class SecurityServerInterceptorFactory
    implements ServerInterceptorFactory {

    private SecurityServerInterceptor   _server = null;
    // cria uma nova intancia do SecurityInterceptor
    public ServerInterceptor create(
        com.visigenic.vbroker.IOP.TaggedProfile profile) {

        if (_server == null)
        {
            _server = new SecurityServerInterceptor();
        }
        return _server;
    }
}
```