UNIVERSIDADE FEDERAL DE SANTA CATARINA
CENTRO TECNOLÓGICO
DEPARTAMENTO DE INFORMÁTICA E DE ESTATÍSTICA
CURSO DE PÓS GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Gerência de Segurança em Aplicações de Bancos de Dados na Web

Alexandre Veloso de Matos
Prof. Dr. Carlos Becker Westphall
Orientador

Dissertação submetida à Universidade Federal de Santa Catarina para obtenção do grau de Mestre em Ciência da Computação.

Florianópolis, março de 1999.

Dedicatória

Muito do que tenho devo a Deus, quase tudo que sou devo ao meu Pai, Sr. Hélio, e tudo o que faço vem de um exemplo materno, da Dona Diléi, a eles dedico estas 104 páginas e estes 28 aninhos.

Gerência de Segurança em

Aplicações de Bancos de Dados na Web

Alexandre Veloso de Matos

MESTRE EM CIÊNCIA DA COMPUTAÇÃO

especialidade **SISTEMAS DE COMPUTAÇÃO** e aprovada em sua forma final pelo **CURSO DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO**.

Prof. Dr. Carlos Becker Westphall
ORIENTADOR

Prof. Dr. Jorge Muniz Barreto
COORDENADOR DO CURSO DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

BANCA EXAMINADORA:

Prof. Dr. Carlos Becker Westphall
PRESIDENTE

Prof. Dr. João Bosco Mangueira Sobral
CO-ORIENTADOR

Prof. Dr. João Bosco da Mota Alves
MEMBRO

Prof. Dr. Luiz Carlos Zancanella
MEMBRO

AGRADECIMENTOS

Meu primeiro agradecimento é a **Deus** !

Mas, também agradeço (e muito) a amigos muito especiais que, obviamente, trouxeram-me e ainda me trazem muita ajuda e alegria, assim como terão sempre o mesmo de mim.

Meu **pai**, minha **família** e a memória de minha querida **mãe** que incentivaram-me na mudança para Florianópolis e ainda me incentivam na minha carreira de maneira esplêndida (como sinto saudades !)

Meus amigos **Ricardo**, **Kormann**, **Taís**, **Maria Laura**, **Bia** e **Mirela**, revisores dedicados e a quem devo muito pelo carinho e reconhecimento (bota na conta que eu pago quando puder !).

Minhas amigas **Tatiana**, **Kátyra**, **Fernanda** e **Karla**, colegas incansáveis na colaboração da elaboração de códigos.

Meus amigos distantes (apenas fisicamente) porém próximos, que souberam me apoiar em todos os momentos: **Cleide** e **Ana Maria**.

Ao meu orientador, pelo paciente incentivo à defesa.

Enfim, a todos que me fizeram entender que era hora de tentar e que tudo podia dar certo mesmo parecendo cedo demais !

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

GLOSSÁRIO DE ABREVIATURAS

API	Aplication Program Interface
BD	Banco de Dados
BOA	Basic Object Adapter
CCA	Carregador de Classes de Applets
CDR	Common Data Representation
CDSA	Common Data Security Architecture
CGI	Common Gateway Interface
CLI	Certificate Library Interface
CLM	Certificate Library Module
COM	Component Object Model
CORBA	Common Object Request Broker Architecture
COSS	Common Object Service Specification
CSP	Criptographic Service Provider
DBA	Database Administrator
DII	Dynamic Invocation Interface
DLI	Data-Storage Library Interface
DLL	Dynamic Link Library
DLM	Data-Storage Library Module
DOK	Distributed Object Kernel
DSA	Digital Signatura Algorithm
ECB	Extension Control Block
GIOP	General Inter-ORB Protocol
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
IAB	Internet Activities Board
IDAPI	Integrated Database Aplication Program Interface
IDL	Interface Definition Language
IIOP	Internet Inter-ORB Protocol
IP	Internet Protocol
IPSEC	Internet Protocol Security
ISA	Internet Server Applications
ISAPI	Internet Server Aplication Program Interface

JRE	Java Runtime Environment
MVJ	Máquina Virtual Java
OA	Object Adapter
ODBC	Open Database Conectivity
OLE	Object Linking and Embedding
OMA	Object Management Architecture
OMG	Object Management Group
ORB	Object Request Broker
OSI	Open Systems Interconnection
PEM	Privacy Enhanced Mail
PGP	Pretty Good Privacy
RMI	Remote Method Invocation
RPC	Remote Procedure Call
SBD	Sistema de Banco de Dados
SET	Securte Eletronic Transaction
SGBD	Sistema Gerenciador de Banco de Dados
S-HTTP	Secure Hypertext Transfer Protocol
SPI	Service Provider Interface
SRM	Security Reference Model
SSH	Secure Shell
SSL	Secure Socket Layer
TCP	Transmission Control Protocol
TPI	Trust Policy Interface
TPM	Trust Policy Module
URL	Uniform Resource Locator
VBC	Verificador de Byte Code
WWW	World Wide Web

RESUMO

O presente trabalho discute os problemas de segurança nas alternativas para o acesso a Bancos de Dados através da Web. Para que o objetivo fosse alcançado, o trabalho está organizado em duas fases distintas:

(1) a discussão de características negativas e positivas de tecnologias utilizadas atualmente e
(2) estabelecimento de condutas de segurança para as alternativas identificadas na fase (1).

Três níveis são identificados na análise da segurança:

(1) nível do sistema operacional, relacionando os principais protocolos de segurança que são adotados como solução a nível de transporte;
(2) nível de segurança do próprio ambiente, ou seja, como os atuais navegadores incorporam segurança e como a segurança oferecida pelos mesmos pode ser útil; e
(3) a segurança que pode ser oferecida por linguagens e arquiteturas como Java e CORBA.

Esta análise permite a obtenção um ambiente onde:

(a) a autenticidade do usuário e do provedor de serviços é verificada;
(b) a confidencialidade de senhas e informações é garantida, isto é, os dados trafegam de maneira confiável;
(c) não repúdio é impedido, ou seja, o usuário e o provedor de serviços não podem desmentir o fato de participarem do processo de consulta, se tiverem participado; e
(d) o controle de acesso é implementado de forma a facilitar mecanismos futuros de auditoria.

ABSTRACT

This work discuss a solution to the security management of sites in Internet that make some access to databases. So, the objective of this work is arranged in two diferent phases:

(1) the discussion of positive and negative behavior of technologies used actually; and

(2) use of security meanings over the options identified in the phase (1).

Three levels are identified in the security analysis:

(1) the Operating Systems level, according to the security protocols adopted as solution to the transport level;

(2) the environment security level. How Internet browsers manage and allow security services;

(3) the security level of the language Java and the architecture CORBA.

This analisys help to obtain an environment where:

(a) the autentication of user and provider is verified;

(b) the confidentiality of passwords and informations is guaranteed;

(c) the nonrepudiation is blocked, allowing that the originator of a message (user or provider) cannot deny that he sent that message; and

(d) auditing mechanisms could be implemented in the future, through an access control.

1 - INTRODUÇÃO

Desde que a Internet passou a ser explorada pelos segmentos comerciais, muitos serviços e aplicações passaram a utilizá-la como infra-estrutura. O motivo é muito simples: a *World Wide Web* é uma imensa vitrine virtual no qual a visita de todos os tipos de clientes possibilitam a divulgação e a venda de produtos e idéias de maneira muito mais rápida e barata.

Neste trabalho, o uso de Banco de Dados como tecnologia para a gerência de informações é discutido, sendo confrontados os problemas relacionados à segurança quando essa tecnologia é exposta à *Web*.

Existem várias aplicações que utilizam a *Web* e também utilizam de Bancos de Dados para fornecer seus serviços. Podem ser citados os seguintes exemplos:

- livrarias virtuais
- supermercados virtuais

BANCOS E ADMINISTRADORAS DE CARTÕES DE CRÉDITO

- jornais e revistas eletrônicos
- empresas para a venda de serviços, tais como turismo, educação, saúde, etc.
- provedoras de acesso à Internet , ...

Nestes exemplos, a informação que está armazenada em seus Bancos de Dados pode conter dados sigilosos, tais como o saldo de uma conta bancária ou os dados pessoais de um cliente de uma provedora. Tecnologias diferentes são adotadas e a escolha de uma solução para a aplicação distribuída deve levar em consideração as alternativas que atualmente existem.

Para o desenvolvimento de aplicações distribuídas, algumas alternativas de implementação são discutidas em [31]. Neste trabalho são confrontadas quatro alternativas:

- uso de CGI (*Common Gateway Interface*) em conjunto com HTML (*Hypertext Markup Language*);
- a tecnologia Microsoft ISAPI (Internet Server Pages);
- uso de Servlets
- e o uso de Java em conjunto com Objetos Distribuídos.

Para a validação da solução, a proposta de implementação da emissão de Extratos de Conta Telefônica através da Web, foi utilizado. Dada a característica do problema, a solução adotada baseou-se no uso de Java em conjunto com Objetos Distribuídos. No entanto, nas respectivas seções onde são tratadas as outras alternativas, são discutidas situações onde as mesmas seriam aplicadas.

Para a inclusão de serviços de segurança em uma aplicação de Banco de Dados na Web, neste trabalho, são feitas considerações em vários níveis em uma estrutura de camadas de uma rede Internet. Ao nível da aplicação, cada alternativa apresenta suas potencialidades e suas fraquezas. Em específico, no capítulo III, os serviços de segurança da arquitetura CORBA e de Java são amplamente discutidos.

Complementando a estruturação dos serviços de segurança, em se tratando do nível de transporte, verifica-se que ambientes tais como *browsers* Internet, oferecem facilidades tais como a possibilidade de se utilizar o protocolo SSL (*Secure Sockets Layer*), ou ainda, SSH (*Secure Shell*), de forma a facilitar a autenticação dos clientes autorizados a utilizar o serviço.

No entanto, neste trabalho, também são feitas considerações sobre outros exemplos de implementação de serviços de segurança. O protocolo HTTPS (*Hypertext Transfer Protocol* sobre *SSL*) é um exemplo de uma extensão ao protocolo HTTP para o estabelecimento de transações WWW seguras, assim como o protocolo SET (*Secure Eletronic Transaction*) é um esforço conjunto da Mastercard, da Netscape e da IBM para o estabelecimento de um protocolo seguro para o transporte de informações em Sistemas de Pagamento Eletrônico (*Eletronic Payment Systems*).

Um problema chave que foi identificado no desenvolvimento deste trabalho foi a adoção de uma infra-estrutura para o Gerenciamento de Chaves. Como a criptografia é um mecanismo de segurança vital para o funcionamento de qualquer serviço de segurança, um esquema de chaves públicas foi utilizado, no entanto, a distribuição dessas chaves levou à necessidade de se criar uma infra-estrutura peculiar que será melhor discutida no capítulo V, onde são descritas as características do sistema para a emissão de Extrato de Conta Telefônica na Web.

O objetivo principal relacionado a este trabalho é prover uma solução relacionada à segurança de aplicações Internet cujas informações são trazidas de Bancos de Dados. Em especial, é discutida a adoção de um *framework* de segurança para aplicações baseadas em Java e CORBA, tecnologias utilizadas na implementação da proposta de validação das idéias deste trabalho.

Outros objetivos específicos também são alcançados:

- no desenvolvimento de um *framework* de segurança, outros ambientes podem contribuir para a implementação de serviços de segurança. O estudo da infra-estrutura fornecida por tais ambientes, tais como um *browser*, ajudam na adoção de uma política de segurança robusta;
- além dessa característica, para manter a privacidade do usuário, ou pelo menos alertá-lo de tal, uma política de privacidade é implementada para a aplicação;
- como conseqüência do estudo para a adoção do *framework* citado, é feita uma análise das tecnologias disponíveis que facilitam o desenvolvimento de serviços de segurança para aplicações Web.

A adoção do *framework* baseou-se nas experiências de outras aplicações distribuídas encontradas na Internet. Tais aplicações possuem as suas características próprias de implementação de seus serviços de segurança e a partir dessas características, o serviço de Extrato de Conta Telefônica na Web recebeu sua estrutura.

Este trabalho está organizado em oito capítulos. O capítulo II relaciona um breve um contexto de trabalhos relacionados a este assunto. No capítulo III, apresenta-se uma rápida discussão sobre Sistemas Distribuídos e sobre o estabelecimento de uma infra-estrutura para o desenvolvimento de aplicações para a Web. No capítulo III, há uma discussão sobre segurança abordando vários aspectos. No capítulo IV, são apresentadas as alternativas discutidas neste trabalho para o acesso a Bancos de Dados através de *sites* na Internet. No capítulo V, através de um estudo de caso, é feita uma validação do estabelecimento de um *framework* envolvendo o uso de Java e CORBA. No capítulo VI, encontram-se conclusões que espelham os resultados esperados e obtidos. No capítulo VII, são apresentadas as referências bibliográficas e no capítulo VIII, está incluso um anexo relacionando parte do código fonte utilizado neste trabalho.

II – TRABALHOS RELACIONADOS

A interseção entre Web e Bancos de Dados tem produzido vários trabalhos não necessariamente relacionados à segurança. Assuntos como o estabelecimento de arquiteturas de integração [13] [14] [47] [48], exibem a preocupação de utilizar plataformas genéricas para o estabelecimento de tal estrutura. Neste contexto, Servidores Web e suas tecnologias envolvidas perfazem uma importante fonte para o fornecimento desse tipo de serviço.

A segurança no acesso a Bancos de Dados através da Web é um assunto bastante relacionado ao estabelecimento de uma infra-estrutura para uma típica aplicação Cliente/Servidor em três camadas. Estudos como os de [39] [49] [50] e [51], relacionam a importância de se controlar os serviços requisitados pelo lado cliente quanto pelas respostas emitidas pelo servidor. Neste caso, participam de um esquema de interação segura a um Banco de Dados na Web tanto as ferramentas e técnicas utilizadas no lado cliente quanto as do lado servidor.

As características dos trabalhos citados acima – estabelecimento de uma infra-estrutura para o acesso a Banco de Dados na Web e a segurança neste acesso, são o assunto deste trabalho. Nosso foco de discussão não vai além das interações no acesso ao Banco de Dados, diferenciando-se assim de trabalhos mais específicos, como os de [47] e [48].

Para atingir-se os objetivos, o estudo de mecanismos e serviços de segurança apropriados foi imprescindível. Em particular, uma alternativa pouco explorada em aplicações atualmente disponíveis – Java/CORBA, mereceu. Considerando o crescimento do uso de Java e a adoção de infra-estrutura suficiente para o uso de protocolos CORBA (tais como IIOP e GIOP) em *browsers* Web, neste trabalho, um exemplo de uso desse conjunto de tecnologias foi usado para validar essa proposta e levantar questões que podem ser utilizadas em um futuro em que ambas técnicas sejam mais utilizadas.

No próximo capítulo, é iniciada uma discussão sobre Sistemas Distribuídos. Essa discussão é importante para que sejam definidas as características que uma aplicação que utiliza da Web para acessar Banco de Dados possui. Após o estabelecimento dos requisitos mínimos desta estrutura, os serviços e mecanismos de segurança são explorados, com o interesse de esclarecer suas potencialidades nas alternativas que serão discutidas nas próximas seções.

III - SISTEMAS DISTRIBUÍDOS

Há vinte anos atrás, Sistemas Distribuídos eram relativamente incomuns. Pouco se discutia sobre como efetuar a distribuição do processamento, mas, com o passar do tempo alguns avanços foram obtidos principalmente com as melhorias em termos de velocidade das redes locais.

[07] define Sistemas Distribuídos como sendo:
"... uma coleção de computadores autônomos ligados por uma rede de computadores e equipado com software distribuído, sendo que o software distribuído habilita a coordenação das atividades destes computadores e o compartilhamento de recursos de hardware e software".

Os Sistemas Distribuídos apesar de consistirem de vários equipamentos e vários recursos, quando bem estabelecidos, passam ao usuário a idéia de se tratar um único ambiente . Aliás, esta transparência é uma característica fortemente desejável.

Segundo [07], alguns requisitos facilitam a identificação e a concepção de um sistema distribuído:

- **transparência**: a possibilidade de esconder do usuário o real local de processamento, fazendo com que o mesmo acredite que tudo se passa localmente. A transparência pode ser alcançada em vários níveis: transparência de acesso, de localização, de falhas, de desempenho, de migração, de replicação e de concorrência;
- **compartilhamento de recursos**: dada a distribuição de recursos em uma rede de computadores, usuários estariam habilitados a compartilhar recursos distribuídos por esta rede;
- **abertura**: permitir que a inovação tecnológica dos recursos seja inserida sem que o sistema seja prejudicado;
- **escalabilidade**: permitir que o sistema cresça sem que o mesmo necessite parar e que o usuário não necessite ser comunicado;

- **tolerância a falhas**: a propriedade de se recuperar de falhas que ocorram, através, por exemplo, de replicação dos recursos; e
- **autonomia cooperativa**: apesar dos elementos componentes serem autônomos, suas computações devem cooperar num objetivo comum.

Uma grande contribuição atribuída ao amadurecimento inevitável dos Sistemas Distribuídos é a possibilidade de utilizar-se da Web para o desenvolvimento de aplicações distribuídas. Uma arquitetura típica dessa arquitetura é baseada nos protocolos de uma rede Internet (TCP, IP, HTTP,...) e é discutida na próxima subseção.

III.1 Arquitetura de Aplicações Web

Documentos requisitados na Web geralmente são endereçados por meio do protocolo HTTP. A interação é muito simples: um cliente, ao requisitar um documento (uma página escrita em HTML, uma imagem, um som, um arquivo, etc.), automaticamente estabelece uma conexão TCP com o servidor de onde requisita-se o documento. A conexão TCP é finalizada assim que os dados são transmitidos.

No entanto, para que alguma interação ocorra, em páginas escritas em HTML podem estar inclusos *applets* escritas em Java, desvios de execução para *gateways* como CGI ou ASP, formulários HTML, scripts em linguagens de descrição tais como JavaScripts, VBScript ou ainda componentes ativos, cuja função é a execução de um código dinamicamente, tais como *ActiveX* e *Plugins*. Tais recursos promovem a abertura e a exposição de informações, tornando clientes e prestadores de serviço vulneráveis a ações maliciosas.

Dessa forma, para que as aplicações desenvolvidas com o intuito de utilizarem a Web não sejam uma porta para ataques, nesse processo de interação, a interface Web (a visão do cliente), o servidor (prestador de serviços da aplicação em questão) assim como o próprio estabelecedor da seção (o protocolo HTTP) devem estar munidos de mecanismos e serviços de segurança. [58] [59]

Por outro lado, Sistemas Distribuídos têm contribuído para o estabelecimento de novos conceitos, importantes para a concepção de novas arquiteturas e paradigmas. Em específico, o conceito de Objetos Distribuídos, uma aliança entre a Orientação a Objetos e a Computação Distribuída é discutido no próximo item.

III.2 - OBJETOS DISTRIBUÍDOS

Segundo [08], Objetos Distribuídos são pontos de inteligência que podem estar vivendo em qualquer lugar em uma rede de computadores. Sua inteligência também reside no fato de que os mesmos publicam suas interfaces de forma a se fazerem presentes e oferecerem o que podem executar para outros objetos, também distribuídos. Dessa forma, operações complexas podem ser distribuídas e serem executadas de uma forma totalmente transparente.

A idéia de objetos distribuídos tornou-se bastante divulgada, e dessa forma, muitas propostas começaram a surgir a ponto de que padrões necessitassem ser impostos. Dois padrões são discutidos atualmente: o padrão proposto pela *Microsoft*, através do COM/OLE e a proposta discutida neste trabalho, da OMG (*Object Management Group*) da plataforma CORBA (*Common Object Request Broker Architecture*).

A idéia de modelos de Objetos Distribuídos transcende a simples arquitetura de computação cliente/servidor. Enquanto mecanismos estáticos como o *RPC (Remote Procedure Call)*, baseados no conhecimento antecipado da localização do serviço fazem a tônica do paradigma cliente/servidor, estruturas mais dinâmicas são utilizadas em CORBA. São os Objetos Distribuídos, cujas interfaces são públicas dentro de um meio e suas tarefas podem ser invocadas por alguma entidade que não conheça a sua localização.

A OMG discute regras no provimento de interoperabilidade de plataformas baseadas em Objetos Distribuídos. Destas discussões foi proposta uma arquitetura geral de referência, cujos elementos desta arquitetura são abstraídos como objetos a fim de que as várias funcionalidades necessárias à

computação distribuída fossem associadas aos objetos. Este modelo conceitual foi denominado *Object Management Architecture* (OMA).

Neste modelo conceituai, seus componentes, que são objetos, são caracterizados como:

- **Objetos de Aplicação**: relacionados às aplicações específicas dos usuários;
- **Objetos de Serviço**: suportam serviços comuns, independentes de domínios de aplicação (COSS - *Common Object Service Specification*); e
- **Ferramentas Comuns**: serviços que são comuns a todos os ambientes onde objetos distribuídos estejam sendo utilizados.

O tópico mais importante deste modelo conceituai (OMA) é a arquitetura CORBA, no qual objetos se comunicam através de um ORB (*Object Request Broker*). O termo *broker* vem do fato de que tal ORB deve fornecer serviços de *brokering*, ou seja, localização de objetos, distribuição de mensagens e a associação de métodos aos objetos corretos.

Apesar de ser considerada uma arquitetura, CORBA transcende sua atuação, sendo considerada por [09] um paradigma de comunicação que se baseia na tecnologia de orientação a objetos para solucionar problemas de computação distribuída.

Além do ORB, a arquitetura CORBA fornece outros serviços como, por exemplo, o BOA (*Basic Object Adapter*), ou seja, uma biblioteca de funções utilizada pelo programa do servidor para localizar e inicializar as implementações e invocar o método apropriado para responder à requisição do cliente, conforme a Figura 1.

Na arquitetura CORBA, existe uma separação explícita entre Cliente e Servidor, ou seja, conforme pode ser verificado na Figura 1. Ao cliente somente é importante conhecer a interface do que precisa para requisitar e o servidor lhe fornece uma implementação para a sua requisição. Ou seja, pelo ORB, somente trafega um tipo de primitiva de comunicação: *request*.

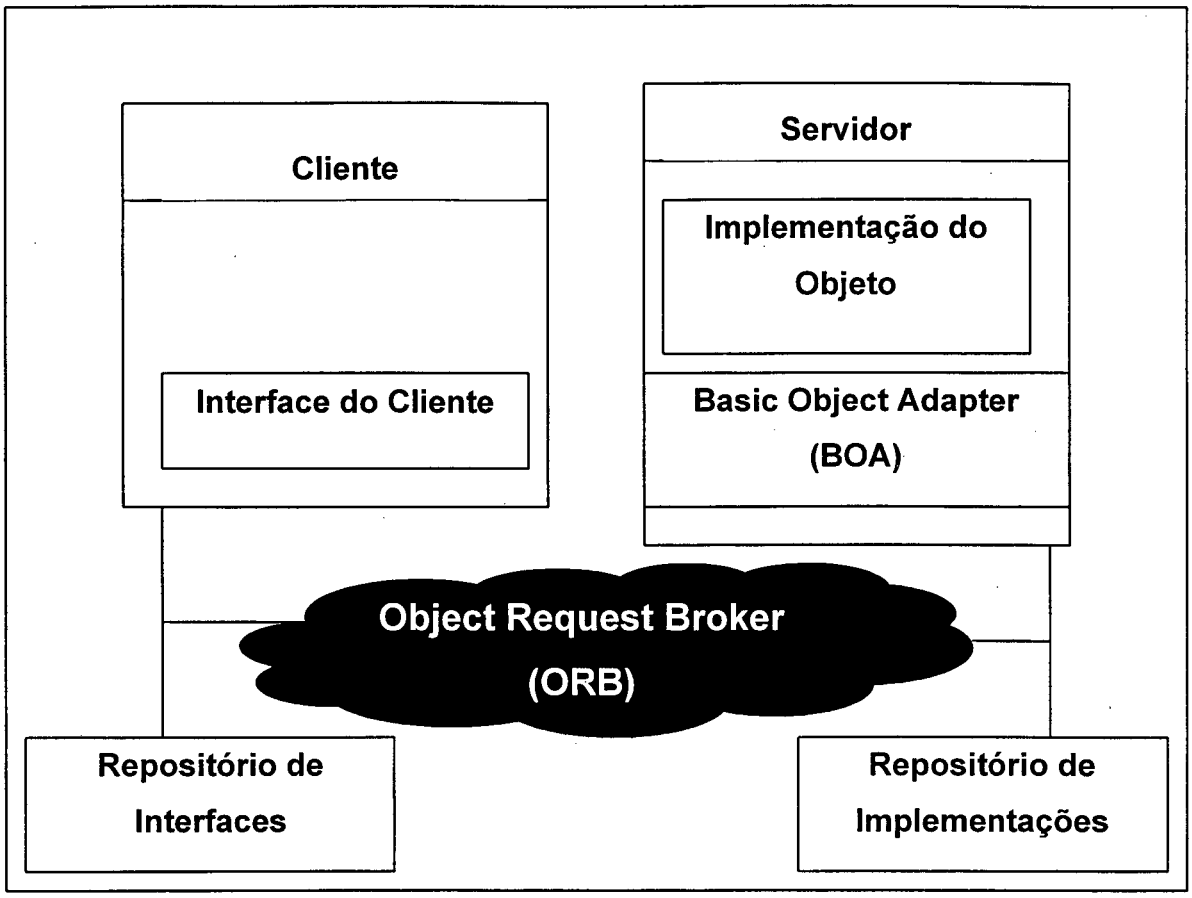


Figura 1 - Visão básica da arquitetura CORBA.

Para que o Repositório de Interfaces seja organizado, um esquema de definição genérica de interfaces deve ser provido. Para tanto, a OMG define uma Linguagem de Definição de Interfaces (IDL). IDL é utilizada para a especificação de todos os serviços que ficarão disponíveis no ORB; trata-se de uma linguagem declarativa que enfatiza a separação entre a interface e a implementação.

Quando uma IDL é compilada, *stubs* são gerados para o cliente e *skeletons* são gerados para o servidor. Os *stubs* representam o mapeamento entre a linguagem de implementação utilizada pelo cliente e o ORB. *Stubs* podem ser vistos como chamadas a procedimentos (não no mesmo sentido de execução que uma RPC), mas, a visão do cliente pode ser esta. Por outro lado, *skeletons* são imprescindíveis para que o ORB e o BOA traduzam a requisição do cliente para um método específico do servidor.

Em acréscimo, para que a implementação de um objeto possa ser invocada, não há a necessidade de se conhecer sua estrutura previamente. Invocações dinâmicas, ao contrário das invocações estáticas representam uma forma pelo qual objetos podem requisitar serviços de outros objetos sem conhecerem suas interfaces previamente, além do que tais objetos podem ser modificados em tempo de execução, pela adição de um novo método, por exemplo.

III.2.1 PROTOCOLOS CORBA PARA INTERAÇÃO DE PLATAFORMAS

O ORB torna-se importante toda vez que o cliente invoca uma operação que tenha sido previamente definida em IDL ou não. A interface disponibilizada através de IDL é o ponto de partida para que outros objetos reconheçam o seu perfil, no entanto, um objeto pode-se comportar tanto como um cliente como um servidor de serviços o que impele à utilização de uma IDL compilada, ou seja, um cliente com seus *stubs* e um servidor com os seus *skeletons*.

Objetos que se comunicam entre ORBs, provavelmente, entre domínios diferentes, precisam padronizar a maneira como as requisições e as respostas são transportadas. Dessa forma, antes mesmo que o pacote seja transportado através de um protocolo como TCP, entra em ação o protocolo GIOP (*General Inter ORB Protocol*) que possui a função de definir uma representação comum dos dados (ocasionalmente referida como CDR - *Common Data Representation*).

A requisição, empacotada pelo protocolo GIOP é passada a uma camada de processamento do transporte de rede, neste caso, representada pelo protocolo IIOP. Esta camada transforma a informação de localização do objeto GIOP em um endereço específico de transporte, sendo que o pacote terá os seguintes campos: *host* TCP + porta para IIOP.

Existem muitas soluções comerciais que utilizam o GIOP e o IIOP como bases para a integração de ORBs CORBA mesmo em domínios distintos. Um exemplo é o *Visibroker Gatekeeper* [12], um recurso que facilita aplicações distribuídas comunicarem-se com outros objetos servidores mesmo através de

um *firewall* sem com isso comprometer a segurança da rede. As tarefas, características e funcionalidades de um *firewall* são discutidos no capítulo IV, seção 1.1.

Para alcançar este comprometimento, duas tarefas são executadas: as de um *firewall* (filtragem de pacotes) e de um servidor HTTP (como atendimento de requisições http). Nesta proposta, o ORB ao ser inicializado é alertado da presença do *Gatekeeper* que somente entra em ação quando alguma requisição não puder ser tratada localmente, neste instante, o próprio *Gatekeeper* passa a ser um procurador do objeto requisitor através, inclusive de outros roteadores, até que o objeto requisitado seja encontrado e posteriormente instanciado.

No capítulo III é apresentado o serviço de segurança CORBA, onde questões relacionadas à interoperabilidade de ORBs é melhor discutida.

III.2.2 - Objetos Distribuídos e Bancos de Dados

Objetos Distribuídos podem ser utilizados em diversos tipos de aplicações. A noção de distribuição do processamento, através de objetos dispostos em locais remotos, alcançáveis a partir de um ORB, permite que os mesmos implementem serviços como o de acesso a dados, por exemplo.

Uma questão fundamental no projeto de Bancos de Dados que se baseiam em Objetos Distribuídos é a granularidade dos objetos, ou seja, qual a complexidade relacionada ao item a ser representado. Em se tratando de Sistemas de Gerenciamento de Bancos de Dados (SGBD) relacionais, podemos ter tabelas, linhas ou até mesmo todo Banco de Dados (BD) mapeado como um objeto. Quando se fala numa granularidade fina, por exemplo, podemos imaginar uma tabela mapeada como um objeto. Conforme cita [09], modificações à sua estrutura podem ocasionar recompilações na estrutura de interfaces e posteriormente na implementação mantida pelo servidor. Neste caso, objetos distribuídos não poderiam ser considerados uma boa opção na distribuição dos dados esperada.

Na verdade, já foi mencionado que a OMG define um esquema dinâmico de associação, chamado de Interface de Invocação Dinâmica (DII), onde a interface do objeto somente é conhecida em tempo de compilação, permitindo assim que modificações estruturais não ocasionem a perda de transparência citada acima.

No entanto, essa flexibilidade gera perda de eficiência na execução da requisição devido à sobrecarga na comunicação cliente-ORB necessária para concluir uma requisição.

Por outro lado, a arquitetura CORBA permite uma certa flexibilidade na implementação dos objetos, o que pode diminuir o *overhead* já citado. Essa flexibilidade parte das características do Adaptador de Objetos (OA) que é o meio pelo qual vários e diferentes tipos de implementações de objetos utilizam o ORB. O OA é uma extensão do BOA e fornece serviços importantes tais como:

- registro de implementações;
- geração e interpretação de referências para objetos;
- mapeamento de referências de objetos para as implementações correspondentes;
- ativação e desativação de implementações correspondentes; e
- invocação de métodos de objetos via *skeleton* ou DII.

Para garantir que tantas tarefas sejam realmente executadas, cada servidor deve possuir vários OAs. Este fato contribui para que as ativações sejam mais facilmente tratadas, o que diminui a sobrecarga de invocações dinâmicas.

Conforme já mencionado, são identificadas duas formas de invocação de objetos: baseada em *stubs* e dinâmica. No primeiro modo, o cliente usa código gerado e identificado em *stubs* e que não pode ser alterado em tempo de execução, o que não ocorre na forma dinâmica. Na verdade, quando um objeto é registrado, é necessário especificar que tipo de política de ativação é utilizada. Essa política identifica como cada implementação é utilizada. Na interface BOA quatro políticas de ativação são verificadas:

- **Servidor Compartilhado**: política na qual um servidor pode suportar mais do que um objeto;
- **Servidor Não Compartilhado**: implementação na qual um servidor suporta apenas um objeto por vez;
- **Servidor Por Método**: um novo servidor é utilizado a cada invocação de método;
- **Servidor Persistente**: é muito similar à política dos servidores compartilhados, no entanto, o servidor nunca é inicializado automaticamente.

Em se tratando de invocação dinâmica, de acordo com [09], além do problema relacionado às modificações da estrutura de objetos, alguns objetos podem necessitar ser acessados de maneira concorrente. Neste caso, duas ações podem ser tomadas:

- uso de *threads* em um servidor cuja política de ativação seja compartilhada;
- uso de servidores separados ativados no modo não compartilhado para cada objeto.

De qualquer forma, o problema de controle de concorrência advindo das necessidades intrínsecas de consultas simultâneas a Bancos de Dados, podem ser gerenciados pelo ORB. Neste caso, se existe mais de uma implementação da mesma interface, usuários podem ser direcionados para interfaces distintas e a consistência deve ser verificada na implementação dos objetos.

CORBA mostra-se uma arquitetura muito útil para o tratamento da heterogeneidade de plataforma e de comunicação. Sua infra-estrutura permite o estabelecimento de uma base interoperável, além de outras vantagens tais como:

- a integração de sistemas de múltiplos Bancos de Dados a outras aplicações que estejam utilizando o ORB; e
- do fato de CORBA ser um padrão, a interoperabilidade pode ser estendida para a comunicação entre ORBs.

A Figura 2 apresenta como é feito o tratamento de uma requisição a um BD, através de um ORB.

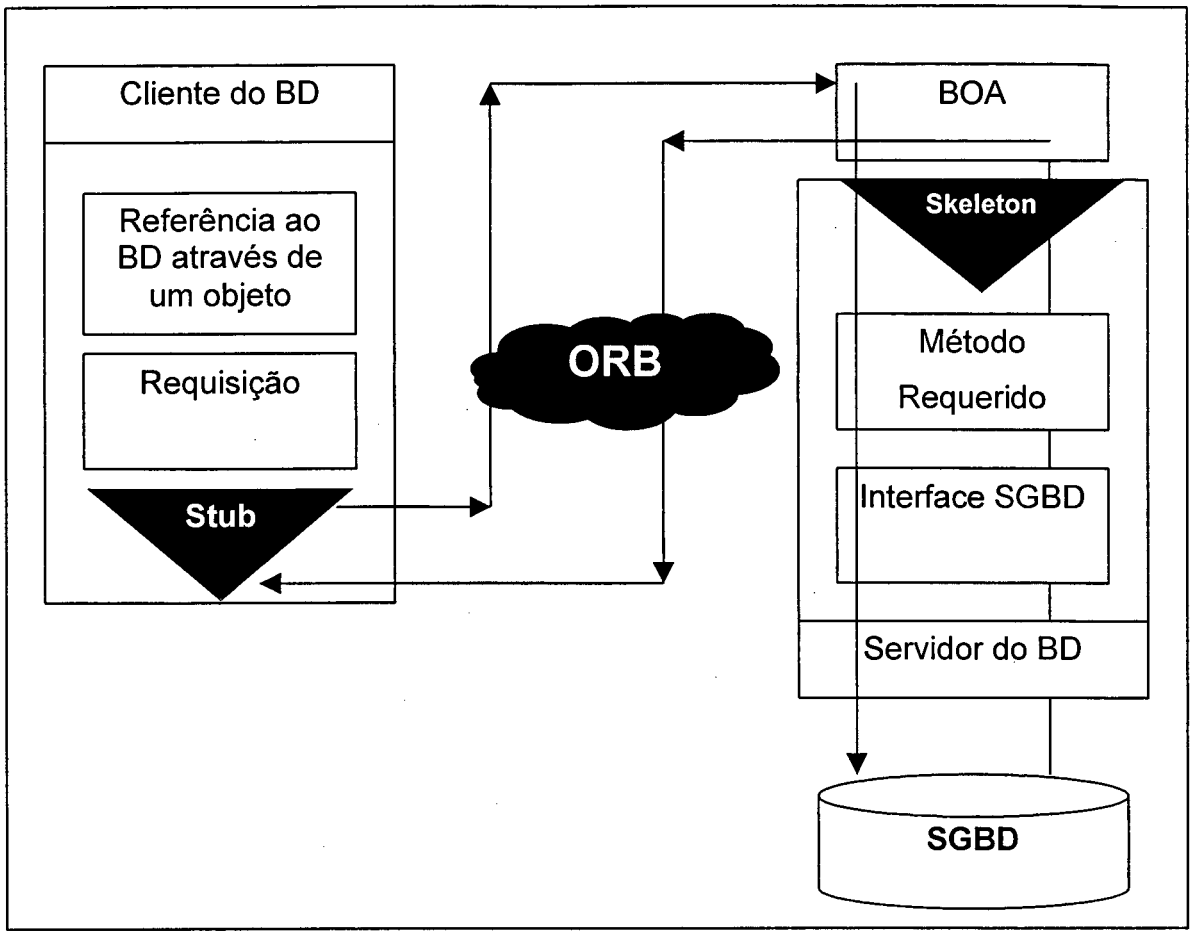


Figura 2 - Requisição a uma instância de objeto de BD através de um ORB.

Em [13] e [14] são reportadas duas experiências no uso de CORBA como infra-estrutura para a utilização de Sistemas de Bancos de Dados pela Web. Na experiência relatada em [14], o projeto DOK (*Distributed Object Kernel*) relata a possibilidade de se construir agentes inteligentes para um acesso seguro a dados na Internet [15]. No próximo capítulo, vários níveis de segurança são discutidos.

IV - Riscos e Serviços de Segurança

Enquanto os computadores eram vistos como equipamentos isolados, a grande preocupação era proteger-se de intrusos no prédio onde o mesmo situava. No entanto, a possibilidade de distribuir informações através de computadores espalhados em uma rede trouxeram inúmeras preocupações, apesar dos ganhos em eficiência.

Portas foram abertas quando mecanismos possibilitaram que uma pessoa em um local distante pudesse conhecer e até manipular o conteúdo de uma máquina remota. Como a confiança, infelizmente, não pode ser totalmente garantida, algumas pessoas descobriram que poderiam intervir em sistemas, prejudicar o processamento de equipamentos remotos e até produzir ameaças perigosas contra um inimigo e lucrar com essa situação.

A abertura discutida por [07] em relação a Sistemas Distribuídos, também permite a ocorrência de situações perigosas para o usuário cuja máquina está conectada a uma rede como a Internet. George Couloris também cita riscos, os quais, quaisquer Sistemas Distribuídos estariam vulneráveis, caso não fosse adotado nenhum mecanismo para garantir a segurança:

- Destruição de informação ou de outros recursos;
- Modificação ou deturpação da informação;
- Roubo, remoção ou perda de informação ou de outros recursos;
- Revelação de informação; e
- Interrupção de serviços.

Em acréscimo, em [16], [17] e [18], os autores discutem riscos que podem ser identificados em sistemas especificamente baseados na Internet. Cameron [16] classifica os riscos de segurança de um sistema computacional como sendo:

- riscos físicos;
- riscos baseados numa rede; e
- riscos baseados nas vulnerabilidades dos softwares.

Quando um risco potencial é intencionalmente efetivado, configura-se um ataque. Um ataque pode ser premeditado através de um cálculo ou de uma inspeção minuciosa das características do ambiente. A maioria dos *hackers*, ou seja, usuários cuja intenção é efetivar um ataque, baseiam-se em vulnerabilidades identificadas no sistema, desta forma, algumas ferramentas, inclusive distribuídas gratuitamente, podem ser utilizadas para explorar tais falhas.

[16] e [19] citam seis aspectos que podem se configurar em ataques numa rede:

- cópia de informações que passam em um roteador;
- pacotes IP com identificação forjada (IP *Spoofing*);
- controle sobre uma conexão autenticada (*Hijacking*);
- vírus;
- sobrecarga deliberada de requisições a um servidor, ocasionando a recusa de serviços; e
- roubo de senhas, através da execução do software *sniffing*.

Além disso, aplicações podem apresentar vulnerabilidades que abrem caminhos para um ataque. De acordo com [16] e [19], podem ser:

riscos no servidor Web

dado o fato de servidores Web proverem informações críticas como: arquivos de senhas e arquivos de páginas, um esforço maior tem de ser concentrado no fornecimento seguro destes serviços. Erros como permissões de arquivos indevidas são extremamente perigosos. Além disso, *scripts* CGI, cujo funcionamento depende da alocação de programas no servidor, podem habilitar o fornecimento de mensagens ou comandos que obstruem o funcionamento do servidor onde o programa CGI se encontra;

riscos no serviço de FTP

o serviço de FTP anônimo é uma porta aberta para a entrada e a saída de arquivos que podem inclusive estar infectados com vírus ou podem

até ser arquivos críticos, ou seja, com informações muito importantes e que não podem ser lidas por qualquer pessoa;

riscos no serviço de E-mail

e-mails podem ser enviados por uma pessoa que se faz passar por outra ou ainda podem ser capturados num roteador e serem redirecionados; e

riscos no serviço de telnet

para se estabelecer uma conexão telnet, senhas e nomes de usuários são fornecidos. Caso os mesmos não estejam criptografados, poderão ser identificados e posteriormente utilizados.

Garantir a segurança em um ambiente aberto é uma tarefa complicada. Algumas premissas podem ser estabelecidas a fim de que a possibilidade de um ataque seja anulada. Em [16] , [17] e [18], os autores relacionam a importância de se estabelecer uma política de segurança, pois desta forma, os serviços empregados poderiam aniquilar tentativas de ataque ou pelo menos diminui-las consideravelmente. Cinco serviços de segurança são a meta da grande maioria das políticas de segurança:

(a) **Controle de Acesso**: permitir o acesso de determinado usuário a recursos que estejam previamente destinados ao uso do mesmo. Dessa forma, apenas usuários autorizados poderiam utilizar o sistema;

(b) **Confidencialidade**: assegurar que a informação na rede mantenha-se privada. Este serviço está relacionado à aplicação do mecanismo de segurança conhecido como criptografia;

(c) **Autenticação**: assegurar que o responsável pelo envio da mensagem é realmente quem ele diz ser. Este serviço está implicitamente relacionado ao uso de certificados digitais;

(d) **Integridade**: assegurar que uma mensagem não tenha sido corrompida ou modificada em seu trânsito; e

(e) **Não Repúdio**: assegurar que o responsável pelo envio de uma mensagem não possa desmentir o fato de que realmente tenha enviado. Trata-se de um serviço muito útil em aplicações comerciais e está implicitamente relacionado a assinaturas digitais.

O projeto de uma política de segurança pode ser mais facilmente organizado se forem previstas as situações de risco, relacionadas a ataques e confrontar os mesmos com os serviços e mecanismos de segurança existentes que facilitem o controle de tais riscos. Na Tabela 1, encontra-se a relação de alguns pontos que podem ser considerados no projeto de uma política de segurança.

Tabela 1 - Considerações no estabelecimento de uma política de segurança.

Riscos	Ataques	Serviços	Mecanismos
Destruição de informação ou de outros recursos	vírus roubo de senhas	Controle de Acesso Confidencialidade	Criptografia Assinatura Digital
Modificação ou deturpação da informação	*IP Spoofing* *Hijacking* monitoração de roteador	Controle de Acesso Confidencialidade Integridade Autenticação Não Repúdio	Criptografia Assinatura Digital Certificados Digitais
Roubo, remoção ou perda de informação ou de outros recursos	roubo de senhas	Controle de Acesso Autenticação	Criptografia Assinatura Digital
Revelação de informação	*IP Spoofing* *Hijacking* monitoração de roteador	Controle de Acesso Confidencialidade Integridade Autenticação Não Repúdio	Criptografia Assinatura Digital Certificados Digitais
Interrupção de serviços	sobrecarga deliberada do servidor	Integridade Não Repúdio	Criptografia Assinatura Digital Certificados Digitais

Uma política de segurança, não deve, necessariamente refletir um nível muito alto de segurança. Os serviços de segurança relacionados somente são aplicáveis se o ambiente realmente está sujeito a riscos.

Atualmente, muitas soluções vêm sendo implementadas e conforme [16], baseando-se em diferentes níveis. Na próxima seção são discutidas soluções relacionadas ao transporte de informações, ou seja, protocolos de segurança utilizados para a implementação de serviços de segurança. Também são discutidas as arquiteturas de segurança providas por Java e CORBA e por último, os recursos de segurança implementados nos *browsers* Internet. O interesse é prover o máximo de informações relacionadas ao ambiente no qual é proposto neste trabalho, cujas tecnologias são: a arquitetura CORBA, a linguagem Java e como interface, *browsers* Web.

III.1 SOLUÇÕES DE SEGURANÇA

III.1.1 PROTOCOLOS DE SEGURANÇA

Em 1994, a *Internet Activities Board* (IAB) divulgou um relatório chamado "*Security in the Internet Architecture*", ressaltando as áreas mais importantes para se implementar mecanismos de segurança. Entre elas foi citada a necessidade de proteção da infra-estrutura de uma rede contra monitoração e controle de tráfego não autorizados, e a necessidade de adicionar técnicas de autenticação e criptografia nessa forma de comunicação digital. As necessidades foram apontadas, porém existia um enorme empecilho: Como adicionar serviços de segurança em uma arquitetura já existente e utilizada? Não havia sentido em desenvolver protocolos super seguros se estes não fossem compatíveis com os utilizados na arquitetura TCP/IP.

As soluções para a garantia da segurança em redes de computadores, dessa forma são aplicáveis a diferentes camadas do Modelo de referência OSI. Enquanto alguns protocolos são implementados junto à camada de aplicação, como é o caso do HTTPS (*Hypertext Transfer Protocol* sobre *SSL*), outros são implementados de forma a complementar o Modelo OSI com uma camada adicional, como é o caso do SSL (*Secure Socket Layer*). Essa diferenciação permite uma maior diversificação de soluções comercializadas. Enquanto uma alternativa tem intenções explícitas de impedir tipos específicos de ataque, outras procuram atender a mecanismos de segurança que possam ser utilizados como a base da construção de uma solução pelo usuário.

Atualmente existem muitos protocolos implementados em diferentes aplicações. Entre estes, podem ser mencionados:

(a) **IPSEC (*IP Security*)** - trata-se de uma atualização dos serviços de segurança relacionados ao protocolo IP. Na versão IPV6, novos serviços de autenticação e criptografia são adicionados, em virtude de ataques como o IP *Spoofing* terem logrado sucesso.

(b) **SSL (*Secure Socket Layer*)** - trata-se de uma proposta mais genérica, onde a sua atuação não está relacionada a nenhuma camada específica. Dessa forma, aplicações desenvolvidas sob este protocolo operam de maneira independente da plataforma e sua característica mais importante é o fato de haver uma preparação minuciosa para a comunicação, desta forma, antes mesmo de ser executada alguma operação relacionada à camada de aplicação, ocorrem a criptografia dos dados e uma combinação dos parâmetros a serem utilizados na comunicação (algoritmo de criptografia, tipo de certificado, etc.).

Vale ressaltar que o protocolo SSL apresenta mecanismos nativos para autenticação de clientes baseados em certificados digitais (assinatura digital), de maneira transparente para a camada de aplicação. No entanto, os mecanismos de autenticação de cliente do SSL não provêem recursos nativos para a implementação de assinaturas digitais sob requisição da aplicação, e consequentemente para não repúdio. No capítulo IV, é discutida uma solução de segurança envolvendo o uso de SSL.

(c) **PGP (*Prety Good Privacy*)** - trata-se de uma aplicação muito popular cuja função é incluir criptografia em sistemas de troca de e-mails. Uma falha é a não centralização da distribuição das chaves públicas, que fica a cargo do proprietário da mesma.

(d) **PEM (*Privacy Enhanced Mail*)** - assim como o exemplo anterior, trata-se de uma aplicação para aplicação de criptografia em e-mails. Diferente da proposta PGP, existe uma autoridade centralizada e hierarquizada para a distribuição de chaves, no entanto, apresenta várias críticas ao algoritmo gerador das chaves (DES - Data Encryption Standard), pois, acredita-se não ser suficientemente seguro.

(e) **SET (*Secure Eletronic Transaction*)** - idealizado como um protocolo de altíssimo nível de segurança, destina-se, basicamente, para o uso em transações comerciais na Internet. Duas empresas administradoras de cartões de crédito, *Visa* e *Mastercard* encampam a implementação do SET.

Por se tratar de um protocolo que visa obter um nível forte de segurança, todos os cinco serviços de segurança citados são implementados: controle de acesso, confidencialidade, autenticação, integridade e não repúdio. Além das características anteriores, assim como o SSL é um protocolo que independe de plataforma e do protocolo de transporte utilizado.

(f) **SSH (*Secure Shell*)** - assim como o SSL, trata-se de uma aplicação de forte autenticação. O SSH pode ser utilizado para garantir que determinadas tarefas comuns sejam feitas de maneira segura e automaticamente criptografadas: telnet, rlogin, rcp e rsh, são algumas tarefas executadas sob o sistema operacional Unix onde determinadas informações como o nome do usuário e sua senha trafegam de maneira limpa ou seja, desprotegidos de uma interceptação. O uso de SSH previne que determinados ataques possam ser bem sucedidos, além de garantir que mesmo que o canal seja inseguro, a comunicação estará sendo executada de maneira segura, pois, a criptografia das informações ocorre antes mesmo da autenticação.

As soluções apresentadas acima relacionam como os protocolos de segurança podem ser utilizados como base para o desenvolvimento de aplicações que requerem segurança dados os riscos na ocorrência de um ataque. Para um desenvolvedor, conhecer tais protocolos é muito importante, no entanto, não se faz necessário aplicar o mais alto grau de segurança em ambientes que onde não há a probabilidade, nem a possibilidade de ocorrerem ataques. Dessa forma, na próxima seção será discutido como o serviço de segurança da arquitetura CORBA pode ser útil para um desenvolvedor que pretende utilizar de tais protocolos de segurança em suas aplicações.

III.1.2 - Serviço de Segurança CORBA

Dada a intrínseca natureza distribuída da arquitetura CORBA, a OMG preocupou-se em especificar um serviço de segurança, projetado para permitir implementações que provenham proteção contra [20]:

- acesso de um usuário autorizado a recursos que deveriam estar escondidos dele;
- a personificação de um usuário com a finalidade de obter acesso a recursos que não poderia utilizar;
- desvio intencional de controles de segurança;
- a interceptação de mensagens em uma comunicação, obtendo-se acesso a informações confidenciais; e
- a modificação de dados interceptados em uma mensagem.

O modelo do Serviço de Segurança CORBA, conhecido como SRM (*Security Reference Model*) tem a finalidade de prover contexto suficiente para que políticas de segurança sejam desenvolvidas de forma a impedir que os riscos acima sejam bem sucedidos. No entanto, devem ser consideradas as particularidades envolvidas nas aplicações que utilizam CORBA.

Mesmo que existam soluções baseadas nos protocolos de segurança já mencionados, níveis de flexibilidade podem ser requeridos nas políticas de segurança. O SRM prevê que algumas características devam ser atendidas, de forma a ser obtida uma política satisfatória. Dessa forma, portabilidade, interoperabilidade, performance e independência tecnológica são aspectos importantes no SRM [20].

- **Portabilidade** - é esperado que o objeto não tenha de se preocupar com a segurança estabelecida no ambiente, pois, dessa forma, o mesmo poderia migrar para ambientes cujo nível de segurança é diferente;
- **Interoperabilidade** - da mesma forma que a premissa anterior, é necessário manter-se a consistência de segurança mesmo entre sistemas heterogêneos, inclusive adotando diferentes ORBs. Muitos ORBs comerciais implementam essa interoperabilidade através do uso de *gateways*, que conectam ambientes distintos. Através do uso dos protocolos GIOP e IIOP já mencionados, objetos podem ser executados e verificados em ambientes de alto grau de segurança, mesmo que não tenha sido implementado com essa finalidade;

- **Performance** - uma questão importante é que a inclusão de serviços de segurança não degrade a performance do sistema, apesar de que um alto nível de segurança, consequentemente cause um *overhead*; e
- **Independência Tecnológica** - para que seja obtida uma independência tecnológica, haja visto que CORBA é uma arquitetura onde os objetos podem ter sido implementados com o uso de linguagens distintas, os mecanismos utilizados devem estar em concordância. Chizmadia [21], lembra que cliente e servidor devem utilizar dos mesmos mecanismos de segurança de forma a permitir o estabelecimento da comunicação.

Dessa forma, percebe-se que a segurança em CORBA pode ser implementada com uma grande variedade de recursos existentes, sendo que mecanismos e protocolos de segurança poderiam ser facilmente adicionados, independendo das características da arquitetura. No entanto, o provimento dos serviços de segurança para o desenvolvedor deveria possibilitar que a política de segurança adotada atendesse aos requisitos enumerados pelo SRM. Na próxima seção é discutida a arquitetura de segurança de Java, utilizada como tecnologia para o provimento de serviços de segurança, no entanto, de acordo com o SRM.

III.1.3 Arquitetura de Segurança Java

Basicamente, Java tem se mantido como uma plataforma bem aceita por um fato simples: ela permite o envio de dados que podem ser automaticamente executados onde quer que cheguem, ou seja, em qualquer lugar da rede, através de pequenas aplicações denominadas *applets*. Esse aspecto é chamado por alguns autores de **conteúdo executável** [01], e por outros, de tecnologia de **código móvel** [02]. Segundo McGraw e Felten [01], documentos que podem ser considerados Conteúdo Executável são aqueles que podem:

- incorporar um *script* de uma linguagem;
- serem transferidos pela rede; e
- serem executados em diferentes máquinas.

Outras tecnologias tais como JavaScript, Active-X, Safe-TCL, Postcript, Macros MS-Excel ou MS-Word e Telescript são alguns dos ambientes competidores para a criação de código executável.

Por outro lado, outras características da linguagem permitem que aplicações convencionais, assim como *applets*, tenham maior facilidade de desenvolvimento:

Portabilidade

A fim de adaptar-se às diferentes plataformas de hardware e software, o compilador Java gera *bytecodes*, cuja função é transportar eficientemente código para ambientes distintos. A portabilidade de Java resume-se à Máquina Virtual Java (MVJ) presente na maioria dos navegadores Internet e responsável pela execução das *applets*.

Flexibilidade

Os programas construídos em Java não precisam ser modificados quando são transferidos de uma máquina para outra. Isso implica na ausência de incompatibilidades entre um programa Java em diferentes ambientes. O fato de Java ser uma linguagem interpretada apresenta uma vantagem a mais, o código de um programa Java não precisa ser recompilado quando é transferido para um ambiente diferente de execução, haja visto que a Máquina Virtual Java pode estar embutida em várias plataformas de execução e reconhecer *bytecodes* produzidos em diferentes compilações.

Gerenciamento de Memória (*Garbage Collector*)

Quando se fala em ponteiros de memória, podem ser imaginadas duas tarefas: alocação e desalocação de memória. A desalocação de memória, quando tratada pelo usuário, pode expor as aplicações a restrições de desempenho e problemas de execução complicados e de difícil detecção.

O modelo de gerenciamento de memória, adotado por Java, não se baseia em ponteiros, mas, em referências a objetos. Quando um objeto não possui mais referências, ele é um ótimo candidato ao *garbage collector*.

Multithreading

Escrever programas onde várias coisas devem ser executadas ao mesmo tempo pode ser complicado. A maioria dos sistemas computacionais possuem um único processador para a execução dos programas, o que pode implicar em esperas intermináveis na combinação de mídias como som e imagem, por exemplo, devido à disputa pelo espaço de execução (o processador).

Multithreading é a base para aplicações construídas em Java. Trata-se de um mecanismo poderoso que promove a performance interativa das aplicações gráficas. Se alguém precisa ter um programa que executa animações e, ainda, emite sons, e se esse alguém ainda precisa executar uma outra tarefa com o mesmo processador, como, por exemplo, a carga de uma página, os processos correspondentes podem ser divididos em processos ditos *mais leves*. Dessa divisão é que surge o termo *thread* (que alguns autores citam como sendo um processo leve, por não ocupar tanto a capacidade do processador, permitindo, assim, um compartilhamento maior do mesmo).

Segurança

A preocupação com segurança é muito maior em aplicações distribuídas e isso se reflete no comportamento padrão da arquitetura de segurança Java: uma *applet* só pode estabelecer ou aceitar uma conexão de rede se for com o seu *host* de origem. Esse tratamento, conhecido como Modelo de Segurança *Sandbox* para Java é muito discutido, devido à sua inflexibilidade, mas, também pelo fato de que muita flexibilização poderia significar brechas para ataques.

III.1.3.1 Componentes da Arquitetura

A fim de tornar o Modelo *Sandbox* menos rígido e para assegurar que tentativas de acesso maliciosas não logrem sucesso, as implementações mais recentes do JDK têm implementado três níveis de verificação para a verificação de aplicações escritas em Java, conforme pode-se identificar na Figura 3.

Verificador de *Bytecode* (VBC)

Antes que seja executado, o resultado da compilação de uma *applet* (*bytecode*) é verificado, de forma a garantir que ponteiros não foram forjados, restrições não tenham sido violadas e que não tenha ocorrido acesso a objetos de informação proibidos.

Carregador de Classes de *Applets* (CCA)

Tendo sido feita a verificação estática do *bytecode*, inicia-se a sua verificação dinâmica. Neste ponto, o CCA determina quando e como uma *applet* pode adicionar classes a um ambiente que esteja executando Java. Para conseguir tal, o CCA divide classes, de acordo com sua proveniência, e associa essas classes com *Espaços de Nomes*. Dessa forma, pretende-se garantir que *applets* maliciosas não tentem modificar conteúdos do ambiente em execução.

Uma forma de otimizar o trabalho de carga de classes em *browsers* Internet é a instanciação de um objeto `ClassLoader`. A classe `ClassLoader` é responsável pela carga normal de *applets* efetuada pelo CCA, no entanto, um objeto instanciada da mesma, pode controlar quais classes são necessárias e efetuar um trabalho sob demanda. No entanto, o Modelo *Sandbox* continua prevalecendo e classes externas ao ambiente continuam impedidas de acessar itens de informação locais.

Gerente de Segurança

É o módulo responsável por verificações em tempo de execução cujas funções são: prevenir da instalação de novos CCAs, proteger *threads* de

interferências mútuas, controlar operações de *read/write*, controlar operações de *socket* (*connect* e *accept*) e controlar o acesso a pacotes Java.

Assim como o CCA, o `SecurityManager` também pode ser instanciado e ainda pode ser responsável por delimitar o *Sandbox*, de forma a permitir que determinadas operações perigosas possam ser executadas, no entanto, sob a permissão do usuário. Essa modificação pode causar transtornos, no entanto, é uma forma de quebrar a rigidez imposta pelo *Sandbox*.

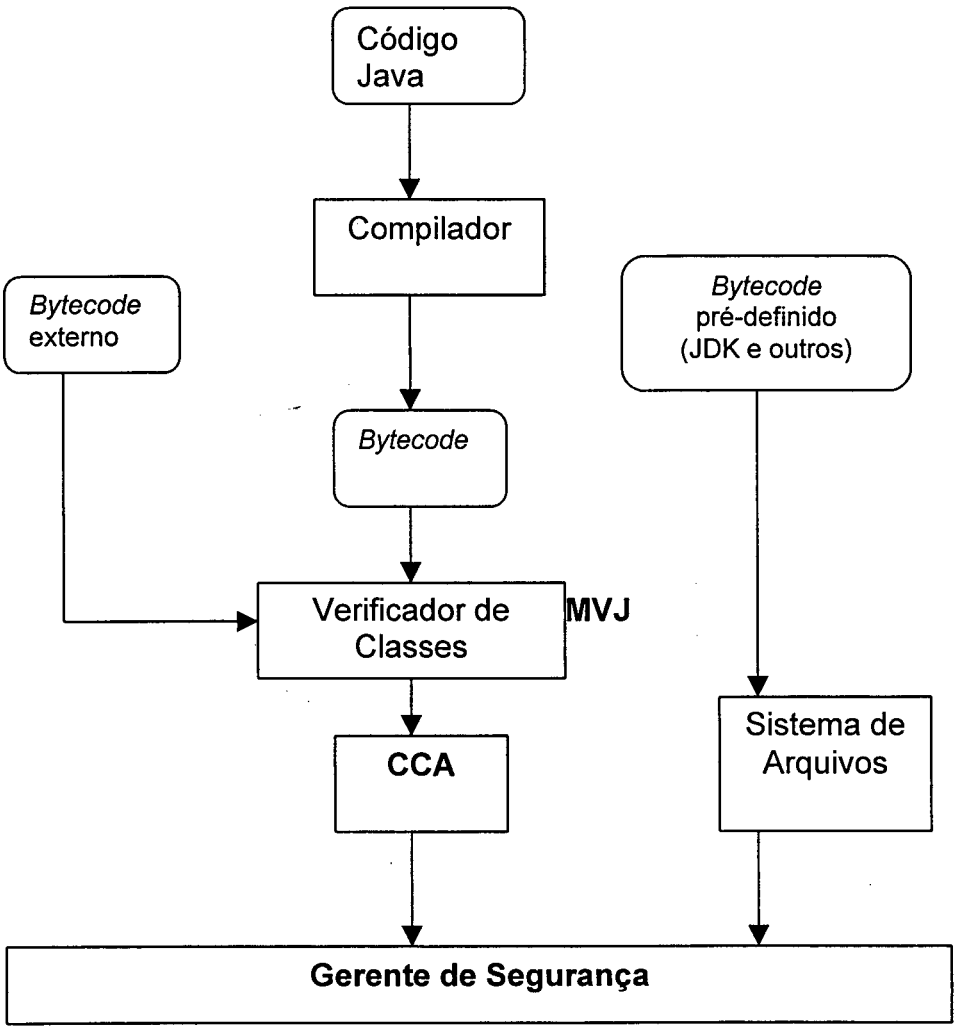


Figura 3 - Arquitetura de Segurança Java.

Dentro do contexto dos três níveis identificados acima, a Máquina Virtual Java possui uma importância muito grande. O fato de estar incluída na grande

maioria de *browsers* ratifica Java como presença importante na Internet. De outra forma, verificações importantes são feitas na MVJ de forma a torná-la também importante para a garantia da segurança, através de [03]:

- verificação de referências a objetos que utilizam classes diferentes;
- acesso estruturado à memória (a não utilização de ponteiros);
- liberação automática de memória (*garbage collection*);
- verificação de limites de vetores; e
- verificação de referências nulas.

Tais verificações efetuadas pela MVJ impedem a possibilidade de ataques cuja intenção é converter a referência a um objeto, ou seja, burlar a segurança dos tipos de forma que um determinado objeto execute operações que não tenham sido definidas para o seu tipo de objeto.

Mesmo com tais níveis de verificação, ataques através de *applets* maliciosas são comprovadamente possíveis [01]. A solução seria controlar as ameaças de ataque através do uso de Java, no entanto, garantindo que a rigidez do Modelo *Sandbox* não fosse um impedimento.

III.1.3.2 Domínios de Proteção

A mais inovadora característica do novo modelo de segurança Java é a implementação de Domínios de Proteção. Em conjunto com outras ferramentas, trata-se de um arquivo texto onde são relatadas as restrições de acesso a determinados itens de informação específicos.

Através dos domínios de proteção [04] [38] [60], a rigidez do Modelo *Sandbox* pode ser amenizado, permitindo que sejam impostas formas específicas de acesso a determinado item de maneira mais flexível. A adoção de um domínio pode ser feita de duas maneiras:

- através da ferramenta `policytool`

- através da API `java.security.Permission` ou qualquer classe `Permission` que seja extensão de algum pacote específico (por exemplo, `java.io.Permission`).

No capítulo V, onde é apresentado um estudo de caso sobre uma implementação que utilize Java e CORBA, a ferramenta `policytool` é melhor descrita.

Java e CORBA são meios importantes para a obtenção de aplicações distribuídas e dinâmicas. No entanto, o perfil dessas aplicações restringe-se à execução em *browsers*. A proposta deste trabalho é poder disponibilizar serviços na Internet cujas informações sejam disponibilizadas a partir de um Banco de Dados, no entanto, esse serviço precisa ser utilizado de maneira segura, portanto, além da Arquitetura de Segurança de Java e do Serviço de Segurança CORBA, a segurança em *browsers* é de importância para o estabelecimento deste ambiente. Na próxima seção, é discutido como são implementados os serviços de segurança em *browsers*.

III.1.4 Segurança em Browsers Internet

O acesso a páginas Web se dá através de *browsers*, uma interface específica cuja capacidade de explorar esse acesso tem possibilitado a migração de vários serviços para essa mídia.

Jornais, revistas, lojas virtuais, pontos de encontro, centros de informação, empresas aéreas, bancos, administradoras de cartões de crédito e uma grande quantidade de empresas tem usado a Web como vitrine de seus produtos e também como ponto de comercialização dos mesmos.

A segurança nesses ambientes, começou a ser discutida quando o interesse de vários segmentos comerciais pelo uso da *World Wide Web* aumentou. Até então, acreditava-se tratar de uma simples vitrine para a exposição de produtos, no entanto, o comércio e a divulgação através da Web pode ser perigoso, caso cuidados com a segurança não sejam estabelecidos.

Para que *browsers* como o *Netscape Communicator* e o *Internet Explorer* possam ser um meio seguro de comercialização e distribuição de serviços, ambos adotam políticas de implementação e disponibilização de serviços de segurança.

O importante é que o usuário possa, ao usar o *browser* como interface de sua aplicação, configurá-la da forma como a sua política de segurança determina. A arquitetura do *Netscape Communicator* [10], permite que *plug-ins* seja anexados, de forma que soluções novas sejam incorporadas sem que o usuário obtenha explicações desagradáveis da impossibilidade de obter o serviço desejado. De outra forma, aplicações consideradas críticas são contempladas com recursos específicos para sua execução. Um exemplo é o comércio eletrônico, que possui estrutura para receber o tratamento de assinaturas e certificados digitais, como descreve a Figura 4.

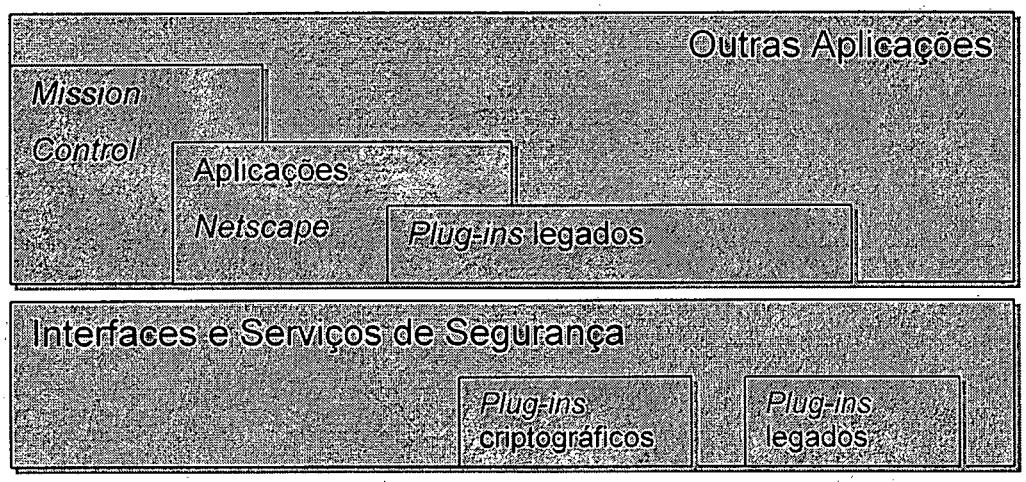


Figura 4 - Arquitetura de Segurança do *Netscape Communicator*.[10]

Além dos *plug-ins,* ou seja, dos serviços adicionais, para mecanismos de criptografia, a arquitetura de segurança *do Netscape Communicator*, possui interfaces e serviços que são utilizados pelas aplicações, assim como *plug-ins* legados por outros sistemas. De outra forma, soluções específicas de uma determinada empresa (neste caso, *Mission Control*), podem utilizar dos recursos de *plug-ins* e também das interfaces e serviços para proverem uma solução.

O *Netscape Communicator* implementa um conjunto de serviços de apoio ao desenvolvedor de aplicações. Conforme se verifica na Figura 5, o CDSA

(*Common Data Security Architecture*) [10] é responsável por disponibilizar o suporte a várias tecnologias relacionadas à implementação de serviços de segurança, de outra forma, possui abertura suficiente para que novas tecnologias sejam incluídas.

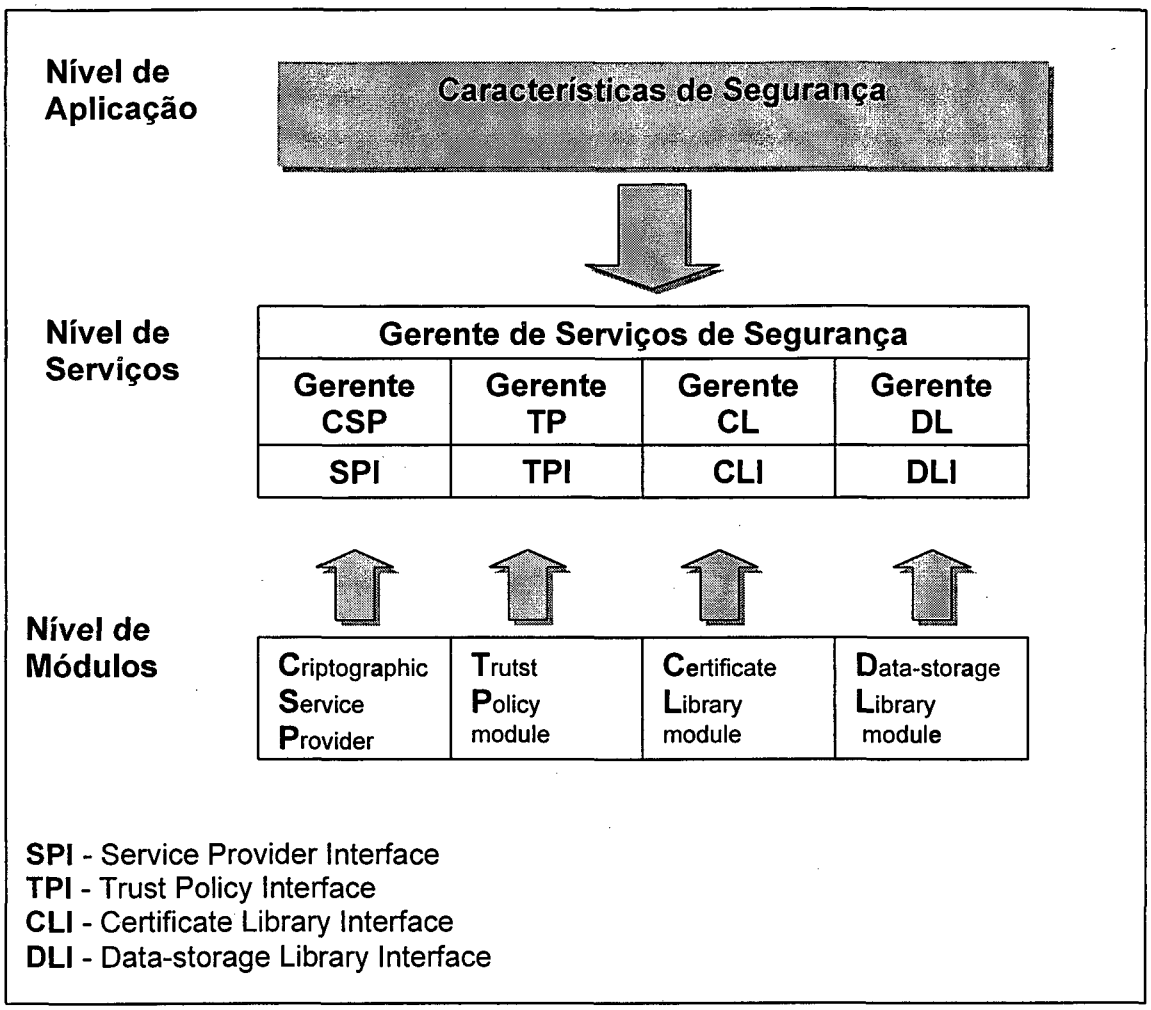


Figura 5 - Arquitetura de Segurança de Dados no *Netscape Communicator*.[10]

A Figura 5 é uma complementação à arquitetura esboçada na Figura 4. As interfaces e serviços de segurança da arquitetura *Netscape Communicator* de segurança podem ser visualizadas em três níveis: aplicação, serviços e módulos.

O nível de aplicação se refere aos serviços de segurança e o protocolo de segurança suportados. Os serviços de portabilidade de chaves criptográficas, assinatura de objetos, criptografia de mensagens, suporte a *Smart Card* e aos recursos do SSL, podem ser encontrados nas atuais versões do *browser*. De

outra forma, algumas APIs precisam ser incluídas de forma a atender os recursos do nível de aplicação. Esse suporte se encontra no nível de serviços, sendo que a API Java é a mais difundida, inclusive em outros *browsers*.

Conforme [10], Java provê aos desenvolvedores a melhor técnica disponível para o desenvolvimento de aplicações independentes de plataforma. Em conjunto com as classes oferecidas pelo próprio Java, outros *plug-ins* podem compor a aplicação desenvolvida pelo usuário, como por exemplo, as facilidades para a criação de uma "plataforma virtual" como a provida pela arquitetura CORBA, através dos ORBs.

Considerada como a mais importante tendência no desenvolvimento de software para Internet/Intranet [10] [22], a arquitetura CORBA é motivo de estudos de melhorias ao CDSA do *Netscape Communicator*. O motivo é que usuários podem acessar aplicações distribuídas através do *Netscape Communicator*. Ou seja, pelo uso de Java ou Javascript em composição com HTML, Objetos Distribuídos podem ser acessados. E mais, pelo uso do protocolo IIOP, a comunicação entre os objetos pode ser gerenciada. No entanto, o protocolo IIOP não é seguro, ou seja, para que comunicações entre objetos sejam seguras, utiliza-se SSL como forma de prover autenticação, confidencialidade, não repúdio, integridade e controle de acesso.

Um outro recurso muito explorado por desenvolvedores são os ***cookies***. São pequenos arquivos texto cuja função é manter informações úteis para a personalização do ambiente do usuário. Em *cookies*, geralmente estão informações sobre o usuário que visita um determinado site, tais como: o domínio de seu provedor, o endereço IP de onde surgiu sua requisição, mas, nunca a identificação pessoal (o *e-mail*) do usuário. São muito utilizados por ferramentas de busca, a fim de informar o usuário de resultados parecidos com a consulta feita assim como fornecer respostas mais próximas de sua realidade. Essa "realidade" é inferida através de seu domínio, por exemplo.

Um arquivo *cookie* não é criado pelo browser, apenas é manipulado por ele. O servidor Web, especificamente o tratador de requisições HTTP possui a

possibilidade de posicioná-lo no disco rígido do usuário. Isso permite ao servidor de páginas identificar que o usuário retornou à página anteriormente visitada, além da função esboçada no parágrafo anterior.

Browsers relacionam-se com *cookies* por serem um intermediário para que o servidor de páginas retorne o resultado mais rapidamente. Além disso, pode-se informar ao browser quais *cookies* são aceitos na máquina do cliente[42] [43].

A relação entre *cookies* e segurança está no fato de que *cookies* podem ser uma boa ferramenta de auditoria. Muitos sites utilizam-no com essa função [44]. De outra forma, *cookies* não podem carregar quaisquer códigos que possam prejudicar o usuário ou executar alguma operação que resulte em descoberta de informações pessoais.

No próximo capítulo, são discutidas alternativas que permitem a *sites* fornecerem informações através do acesso a Bancos de Dados. As considerações efetuadas neste capítulo são muito importantes para que seja concluída a plataforma ideal, no que diz respeito à segurança.

IV - ALTERNATIVAS PARA O ACESSO A BDS ATRAVÉS DA WEB

Neste capítulo são relatadas e analisadas quatro possibilidade para que *sites* possam fornecer informações através do acesso a Bancos de Dados. O propósito é especificar uma plataforma ideal que será utilizada no *framework* proposto neste trabalho.

São avaliadas as seguintes alternativas:

- uso de CGI em conjunto com páginas HTML;
- uso da tecnologia ISAPI;
- uso Java em conjunto com CORBA; e
- uso de *servlets* Java.

Através de códigos de exemplo são discutidas as opções e apresentadas vantagens e desvantagens verificadas nas mesmas.

IV.1 - CGI/HTML

Para que páginas Web possuam alguma interatividade, *scripts* CGI e formatações em *HyperText Markup Language* (HTML) podem ser utilizadas. HTML é uma linguagem de formatação, cujo conteúdo é interpretado por *browsers* de forma a exibir documentos WWW. Ou seja, o mecanismo CGI pode ser visto como composto de:

- um *script*, identificado em *tags* incorporadas em documentos HTML e
- um programa executado no servidor WWW.

Na verdade, programas CGI não possuem nenhuma funcionalidade sem a existência de um formulário escrito em HTML.

A execução de alguma requisição expressa num *script* CGI possui três passos:

(1) a partir de uma página Web, uma aplicação inicia-se quando um formulário HTML recebe o clique do usuário em um botão que **(2)** desencadeia uma chamada a um programa específico que está localizado no servidor WWW. **(3)** Após o *script* CGI ser executado, os dados de retorno são novamente gerados em um formulário baseado em HTML.

A Figura 6 apresenta como são feitas essas interações.

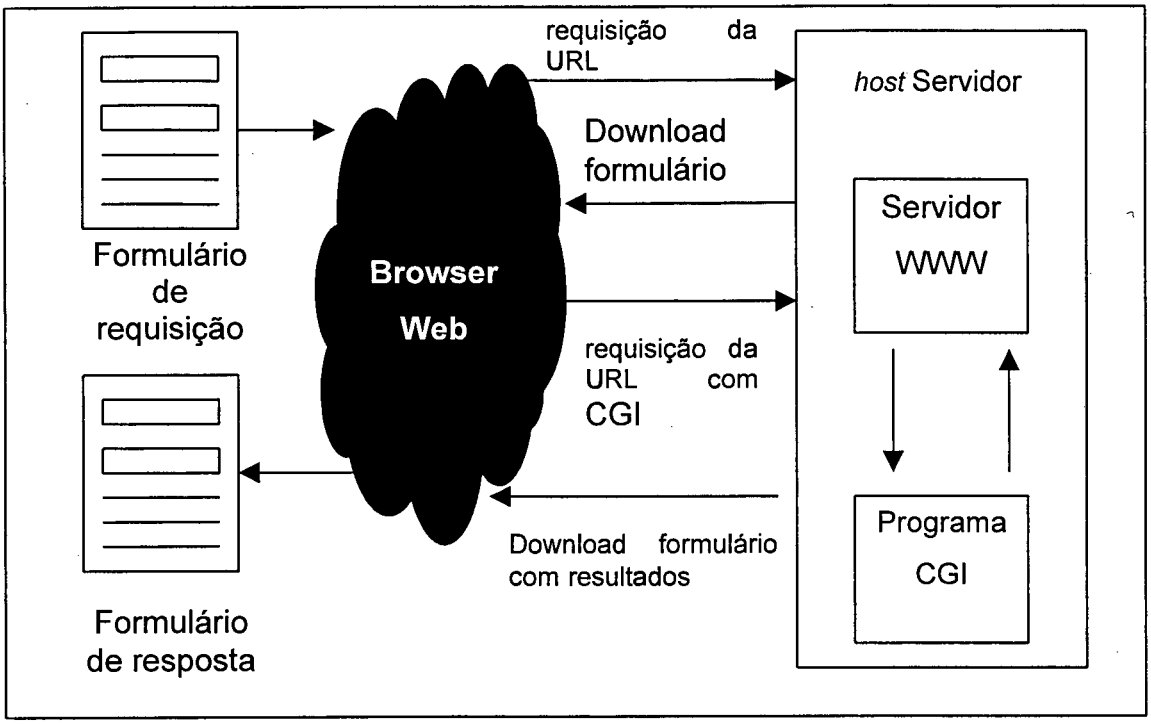


Figura 6 - Visão de uma requisição CGI.

Os programas CGI que residem no servidor WWW devem ser escritos em uma linguagem que consiga tratar e gerar dinamicamente páginas HTML. Esse é um ponto positivo, já que várias linguagens, tais como Perl, C, VBScript, o shell do Unix e Clarion, têm essa característica, tornando a opção relativamente independente de plataforma. Na Figura 7, é apresentado um trecho de código incluso em uma página HTML onde é feita uma chamada a um programa CGI.

```
<HTML>
<HEAD>

<TITLE>Extrato de Conta Telefonica </TITLE>

</HEAD>

<BODY>
<TD><FORM action="/cgi-win/autentica.exe" method="Post">

...
```

Figura 7 - Invocação de um programa CGI em uma página HTML.

A Figura 7, implicitamente revela algumas características desta técnica:

- programas CGI residem em um local denominado *cgi-bin* na máquina servidora;
- programas CGI são invocados por uma *tag* simples da linguagem HTML: <FORM action... > que pode ser utilizada para outras funções;
- method relacionado na *tag* FORM está relacionado à maneira como o protocolo HTTP manipula e organiza os dados, que deverão ser fornecidos por outra *tag* de HTML <INPUT ...>.

O método ao qual se refere, é utilizado para a submissão de valores ao programa que tratará a requisição. O método POST, indicado na Figura 7, trata as informações enviadas para o servidor como parte da mensagem, de outra forma, um outro método, como o GET, trataria as informações como parte de uma URL (*Uniform Resource Locator*), inclusive sendo possível conhecer-se esses valores através da variável de ambiente QUERY_STRING. Ou seja, caso não tenha sido utilizado o método adequado, informações como a senha de um usuário poderiam ser facilmente identificadas [24] [25].

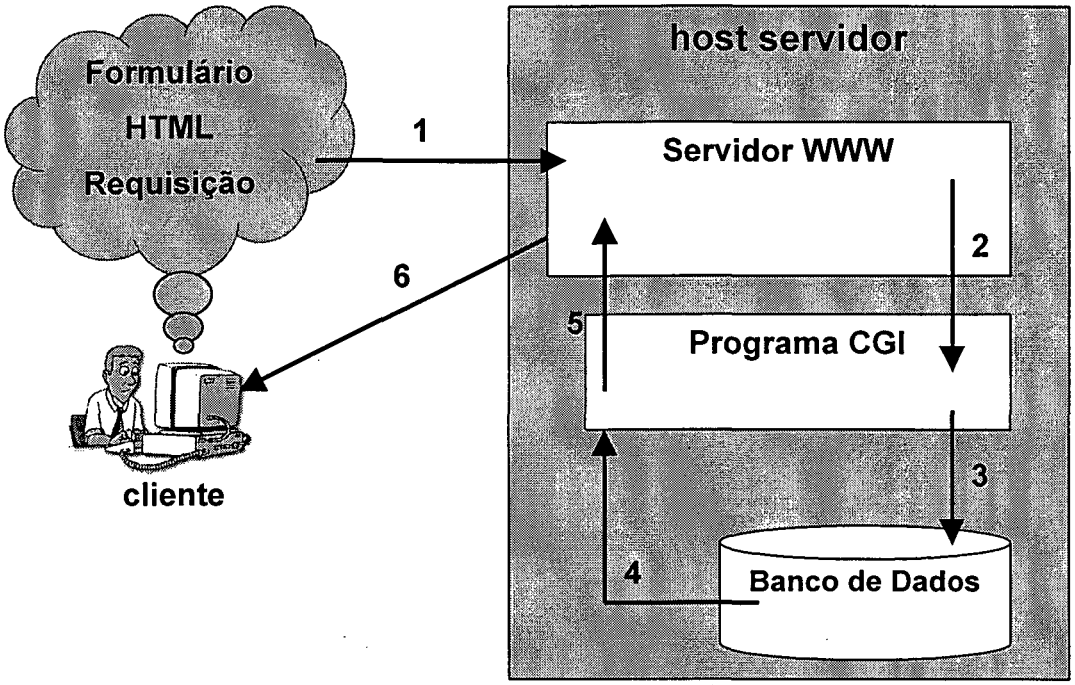
Diversas vulnerabilidades no uso desta alternativa já foram detectadas [16] [23] [24] [25] [26], o que impele à necessidade de um cuidado maior no que diz respeito à segurança. Na verdade, muitos riscos podem existir devido à inexperiência de usuários, mas, outros podem ser atingidos com a intenção explícita de um ataque. Por exemplo, a Figura 8, exibe como um ataque no qual o arquivo de senhas de um determinado *site* é corrompido.

```
<HTML>
    <BODY>
     <H1> Requisição de formulário </H1>
     <FORM  ACTION="http://www.algum.lugar.com.br/cgi-
bin/resposta.exe" METHOD="get">
     <INPUT TYPE="hidden" NAME="meu_endereco"
            VALUE="; rm *; mail -s pessoa@no.mundo.com.br
<etc/passwd;">
        <INPUT TYPE="text"      NAME="comentario">
        <INPUT TYPE="submit" VALUE="Enviar para pessoa">
     </FORM>
   </BODY>
  </HTML>
```

Figura 8 - Exemplo de como invadir um *site* a partir de um formulário CGI.

O trecho de ação do formulário acima requer o conhecimento prévio da função do programa CGI resposta.exe. O usuário `pessoa@no.mundo.com.br` tem conhecimento de que o programa resposta.exe envia *e-mails* para um determinado endereço, desta forma, este programa pode ser manipulado, de forma que seja enviado um *e-mail* para o usuário pessoa e ainda, com o arquivo de senhas do site em anexo.

O uso de CGI/HTML para o acesso a Banco de Dados se dá conforme a Figura 9. Neste esquema, o Banco de Dados é local, ou seja, permanece no mesmo *host* onde o servidor HTTP está. Essa característica é uma limitação, pois, serviços diferenciados não podem ser fornecidos por outros Bancos de Dados, além do fato de que exista a imposição de uma localização específica para o Banco de Dados, não podendo ser utilizado outro *host* que tivesse maior capacidade de armazenamento, por exemplo.



1 - requisição da URL de execução do programa CGI;
2 - localização do programa CGI ;
3 - desvio para um Banco de Dados local para o retorno de dados;
4 - retorno de dados ao programa CGI;
5 - o programa CGI produz o formulário de resposta e o direciona ao servidor que sabe para quem retornar; e
6 - o formulário com a resposta é retornada ao usuário.

Figura 9 - Requisição a um Banco de Dados através de CGI/HTML.

Além disso, outras questões relativas à performance podem ser discutidas. Em [23], o autor lembra que por se tratar de uma solução estática, no sentido de que os programas servidores necessitam estar em um local pré-definido, não são verificadas as mesmas vantagens de uma técnica baseada em Objetos Distribuídos, ou seja, no caso de falhas, o serviço não poderia ser direcionado para outro servidor, além de haver uma maior degradação da performance dado o fato de que a invocação ao programa precisa ser tratada a cada nova chamada, até mesmo no envio da resposta ao usuário.

Como o trabalho do servidor, nesta alternativa, é bastante utilizado, a implementação de serviços de segurança para essa alternativa exige o estabelecimento de primitivas de segurança no servidor.

O conceito de Servidores Seguros é bastante utilizado pelas Autoridades de Certificação, responsáveis pela emissão de certificados digitais que comprovem a veracidade de alguma chave pública, consequentemente comprovando a identidade de uma pessoa ou entidade.

No Brasil, a CertiSign [61] é um exemplo de uma autoridade certificadora. O papel de tais autoridades é prover confiança no fornecimento de chaves públicas, permitindo dessa forma que as comunicações que sejam estabelecidas com servidores Web possam ocorrer de maneira cifrada, impedindo, ou dificultando em muito a interceptação de mensagens.

O protocolo SSL é bastante utilizado, sendo comum utilizar-se o termo HTTPS para representar o fato de utilizar-se dos serviços do protocolo SSL sobre o HTTP. Esta é a tecnologia discutida neste trabalho para propor uma solução de segurança para esta alternativa.

IV.1.1 Arquitetura de uma aplicação Web HTTPS

Três entidades estão envolvidas no desenvolvimento de uma aplicação Web que utilize o protocolo HTTPS: [62] [63]

- a interface usado pelo lado cliente
- prestador de serviços (servidor)
- algum dispositivo de controle de segurança, por exemplo, *firewalls*.

O cliente precisa de uma interface que possa estabelecer um canal de comunicação que utilize SSL, neste caso, *browsers* podem ser utilizados, pois, em sua grande maioria, os mesmos possuem suporte para tal.

Deve ser detectado pelo servidor da aplicação a presença deste recurso, de forma a que toda comunicação estabelecida seja transportada de maneira cifrada.

O servidor, ao gerar o resultado de uma consulta (geralmente através de páginas dinâmicas - resultado da implementação da camada de apresentação) também deve assegurar a integridade e a segurança das informações fornecidas. Neste caso, três ações são necessárias [64]:

- obter o contexto da requisição (controlar a sessão);
- verificar as permissões de acesso;
- efetivamente, executar o serviço pedido.

Utilizando-se dessa alternativa (CGI/HTML), assim como ISAPI ou servlets, as funções do sistema somente são executadas pelo servidor, no entanto, a fim de controlar o tráfego de informações que são enderaçadas ao servidor, evitando ameaças comuns a ambientes distribuídos, um *firewall* pode ser utilizado.

Um firewall, possui, basicamente, três características de projeto: [65]

- monitorar todo tráfego que entra e que sai de um ambiente de rede, ou seja, todo acesso à rede local é fisicamente bloqueado pelo *firewall*, dessa forma, tais pacotes devem passar por este dispositivo;
- implementar uma política de segurança que assegura apenas o tráfego autorizado;
- próprio *firewall* é imune à penetração.

Um *firewall* possui, a princípio, a primária tarefa de controlar os serviços prestados pelo servidor, garantindo que os ataques elencados na tabela 1 não tenham sucesso, no entanto, conforme [65], outros controles podem ser efetuados por um *firewall*:

- **controle do serviço**: limitação dos tipos de serviço Internet que podem ser acessados, interna e externamente. O filtro é baseado no endereço IP e na porta TCP. Um software, denominado *proxy* é o responsável por receber e interpretar cada requisição de serviço e direcionar a decisão pela execução (é um controle muito útil para que o servidor Web apenas execute aquilo que não o prejudique);

- **controle de direcionamento**: podem ser estabelecidas as inicializações que um determinado serviço pode ter para que requisições particulares (previamente estipuladas) sejam executadas;
- **controle de usuário**: geralmente o controle de usuário é feito de maneira local, no entanto, o acesso de usuários externos também pode ser controlado, caso seja executado em conjunto com outro protocolo. Neste caso, como *firewalls* baseiam-se no controle a partir de um determinado IP, pode ser utilizado o protocolo IPSec;
- **controle de comportamento**: controla o comportamento aceitado por determinado serviço, dessa forma, um serviço de e-mail, por exemplo, pode ser avisado de uma tentativa de *spam* (uma espécide de sobrecarga deliberada do servidor, conforme a Tabela 1), ou até o sistema pode ser prevenido de *applets* maliciosas serem executadas. [66]

Numa alternativa onde páginas dinâmicas são geradas a nível da camada de apresentação para esboçar os resultados de uma consulta a um Banco de Dados, tais como CGI/HTML, ISAPI, Servlets e outras não citadas neste trabalho, como, por exemplo ASP, o uso de um *firewall* libera muitas verificações que deveriam ser feitas pelo servidor. Muitas empresas (a grande maioria) que disponibilizam serviços através da Web utilizam de *firewalls*, inclusive para proteção contra os usuários internos.

Sobre essa alternativa podem ser verificadas muitas características negativas relacionadas à performance, assim como com relação à performance. No entanto, a adoção de posturas tais como o uso de *firewalls* em complemento com o uso de HTTPS podem torná-la uma solução atrativa, caso a complexidade do problema seja simples. Conforme será discutido na seção 1.4, quando se trata de uma solução totalmente distribuída, essa não seria uma boa solução para o acesso a Banco de Dados. Na Tabela 2, portanto, são exibidas as características positivas e negativas dessa opção.

Tabela 2 - Vantagens e desvantagens da proposta CGI/HTML.

CGI/HTML	
Pontos Positivos	**Pontos Negativos**
há uma relativa independência de plataforma, pois na verdade, um programa escrito em shell Unix não seria executado em um servidor Windows NT, por exemplo;	há vulnerabilidades em relação à segurança já verificadas;
há confiabilidade no seu uso, mas, apenas se existir uma política de segurança rígida;	possui um intrínseco "gargalo" de performance devido à sua política de acesso;
não é aplicável apenas à consulta a Bancos de Dados, pode ser mais conveniente em rápidos processamentos, como o cadastro em listas.	é uma técnica centralizada, não possuindo as mesmas vantagens que Objetos Distribuídos poderiam fornecer quanto à distribuição do processamento.

IV.2 - ISAPI

O desenvolvimento de aplicações para a Web caracteriza-se pela estruturação dada aos serviços. A proposta ISAPI, discutida nesta seção, estabelece um conjunto de técnicas para que aplicações servidoras na Internet (ISA - *Internet Server Applications*) sejam construídas.

A motivação para a adoção da opção ISAPI advém de um fator negativo identificado em alguns servidores HTTP. Tais servidores ao tratarem as requisições, criam processos separados para cada uma. Dessa forma, uma fila de requisições significará uma fila de processos, o que pode incorrer em um processo de degradação de performance do sistema.

Essa falha pode ser verificada em relação às aplicações CGI. Conforme já discutido, a cada requisição, um processo para tratar o pedido é criado e dessa forma, a alternativa CGI incorre em problemas de performance.

Uma forma de corrigir essa falha é permitir que os processos somente sejam criados quando se tornar necessário, ou seja, se uma aplicação já tiver sido

requisitada, não criar novamente um novo processo para tratá-la. Essa proposta é adota pelo *Internet Server API* (ISAPI) [28].

Utilizando ISAPI, a requisição é direcionada a um arquivo DLL (*Dynamic Link Library*), cujas funções mantêm-se ativas na memória do servidor, permitindo dessa forma que não seja necessário estabelecer um novo processo, caso a DLL já tenho sido carregada em memória.

A arquitetura de aplicações ISAPI estabelece uma unidade de comunicação entre a aplicação e o servidor, são os ECB (*Extension Control Block*) cuja função é informar ao servidor características da requisição do usuário e livrá-lo de ter que monitorar informações sobre o pedido, tais como o *status* da transação feita pelo usuário. A Figura 10 exibe como são feitas essas interações em comparação à alternativa CGI.

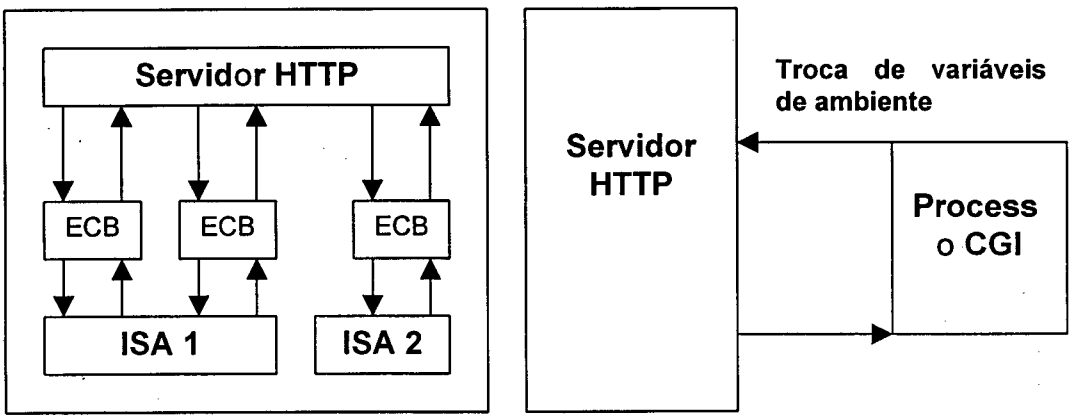


Figura 10 - Arquitetura de aplicações ISAPI em comparação a CGI/HTML.

De acordo com a Figura 10, percebe-se que o uso de ECBs permite que sejam executadas respostas dinâmicas às requisições HTTP, ao invés de retornar uma página estática em HTML. Essa característica permite que a construção de uma requisição de informação armazenada em um Banco de Dados no servidor local tenha um tempo de resposta muito melhor que a proposta CGI/HTML, no entanto, esbarra no mesmo problema identificado na proposta anterior: o Banco de Dados necessita permanecer no servidor local, ou seja, não haveria possibilidade de distribuir os dados, ou alocá-los em *hosts* diferentes.

Além da característica citada acima, o uso de ISAPI é muito similar ao de CGI, conforme identifica a Figura 11. A execução de uma requisição ISAPI se dá através de servidores baseados em Windows NT. Dessa forma, tem-se uma proposta bem direcionada, aplicável apenas a um ambiente específico.

```
<html>
<title>Exemplo de ISAPI: Pesquisa simples de texto</title>
<BODY BACKGROUND="/samples/images/backgrnd.gif" BGCOLOR="FFFFFF">
<TABLE>
<tr>
<TD>
<font size=2>
<h1>Pesquisa simples de texto</h1>

<form METHOD="POST" ACTION="/scripts/samples/srch.dll">
Digite o texto a ser procurado:<br>
<input text name="input_text">
<input type="submit" value="Pesquisar">
</form></font></td></tr></table></body></html>
```

Figura 11 - Código HTML com uma requisição ISAPI.

Assim como CGI, trata-se de uma alternativa cujo produto final é o fornecimento de uma página dinâmica. Conforme discutido na seção anterior, prover soluções de segurança para esse ambiente requisita o uso de técnicas e tecnologias tais como: HTTPS e *firewalls*. No entanto, em se tratando de performance, trata-se uma solução mais agradável, diminuindo a quantidade de interações necessárias para o estabelecimento de alguma reposta. Na Tabela 3, portanto, estão expressas as características positivas e negativas dessa proposta.

Tabela 3 - Quadro comparativo da proposta ISAPI.

ISAPI	
Pontos Positivos	**Pontos Negativos**
possui uma relação de performance melhor que a alternativa CGI/HTML;	solução proprietária;
boa alternativa para a geração de páginas dinamicamente;	pelo fato dos ECBs possuirem informações vitais para o tratamento de requisições pelo servidor, quaisquer violações transportadas pelos ECBs podem ocasionar a falha geral do servidor;
o servidor HTTP não precisa se preocupar com todas as	é uma técnica centralizada, não possuindo as mesmas vantagens que

características da requisição. Os ECBs guardam os *status* e permitem o servidor tomar decisões rapidamente.	Objetos Distribuídos poderiam fornecer quanto à distribuição do processamento.

IV.3 - SERVLETS

A API Java *servlet* é um recurso adicional para os desenvolvedores de aplicações Web. Trata-se de um mecanismo simples cuja intenção é, basicamente, incrementar a funcionalidade de um servidor Web.

Em 1997, a Sun MicroSystems lançou o *Jeeves* [69] [70], um servidor Web baseado em Java. Junto com o lançamento deste produto, percebeu-se a necessidade de prover-se um dispositivo semelhante a CGI e ISAPI, um gerador de páginas dinâmicas, ou seja, um gateway para o tratamento de requisições HTTP que não fossem páginas, mas, programas. Dessa forma, surgiram os *Servlets*, uma espécie de extensão das *applets* [31] [33] [34]. Enquanto *applets* são uma aplicação dirigida ao usuário no lado cliente, *servlets*, comunicam-se com o servidor, produzindo resultados parecidos com os observados nas propostas CGI/HTML e ISAPI.

Outra diferença entre *applets* e *servlets* está no escopo de operações permitidas as ambas. Enquanto *applets* apenas podem efetuar escrita de arquivos e abrir *sockets* mediante explícitas permissões, *servlets* possuem uma maior flexibilidade para obter tais operações no servidor. De outra forma, seu ambiente de execução não seriam os *browsers* Web e sim um ambiente de execução específico no servidor.

Servlets são executadas a partir de uma requisição expressa em um *browser* Web ou até mesmo por uma aplicação que acesse o servidor. Essa requisição é repassada ao *Servlet Engine*, um ambiente para execução de *servlets* no servidor. A resposta é devolvida através do protocolo HTTP.

A caracterização da execução de um *servlet* o coloca como uma aplicação intermediária entre CGI e ISAPI. No entanto, com a vantagem explícita de não se tratar de uma opção proprietária e de outra forma, por permitir que as

requisições possam ser tratadas em *threads* diferentes, tornam-na uma solução mais rápida.

O ciclo de vida de um *servlet* determina quatro momentos [31]: armazenamento, inicialização, recepção das respostas e destruição. No instante da inicialização, muitas tarefas que serão executadas podem ser preparadas. Por exemplo, podem ser carregados os drivers JDBC para o acesso a Bancos de Dados.

O que diferencia as implementações de *servlets* são as diferentes versões do *Engine Servlet*, ou seja, dos responsáveis por executar a *servlet* no servidor. Existem muitas implementações que tornam a execução de um *servlet* muito dependente da característica do *Engine Servlet* [34]. De outra forma, um recurso de difícil utilização nesta proposta são os Objetos Distribuídos. Apesar de algumas soluções terem sido propostas [36], a própria característica de utilização dos *servlets* dificultam a distribuição.

Servlets são, em primeira instância, uma contrapartida para as limitações percebidas na alternativa CGI/HTML. Ou seja, um Banco de Dados que seja utilizado por essa proposta estaria localizado no *host* servidor, ou seja, não seria permitido, dessa forma, a adoção de uma política real de distribuição.

A primeira solução a ser adotada poderia ser o uso de RMI (*Remote Method Invocation*) em conjunto com *servlets* [36], no entanto, duas questões inviabilizariam essa prática:

- uso de RMI, ou seja, uma alternativa puramente Java para a distribuição de objetos necessita da inclusão de um nível a mais de complexidade: um *driver* para a conversão de objetos não escritos em Java (conforme se verifica na próxima seção); e
- conforme pode ser verificado em [35], *servlets* são aplicações complexas, sendo aconselhável utilizar somente quando se fizer necessário, ou seja, se outra proposta puder ser utilizada, mas, que garanta performance, a mesma seria mais válida.

Na verdade, três pontos positivos podem ser destacado em relação à alternativa baseada em *servlets*:

- apesar de existirem implementações diferentes para *Engine Servlets*, é possível criar-se uma servlet para inicialização do *Engine Servlet* específico; [34];
- *servlets* são aplicações que podem executar coisas que somente são permitidas a *applets* mediante o estabelecimento de uma política de segurança. Por exemplo, a escrita em arquivos e a abertura de *sockets* não seriam tão problemáticas;
- atualmente, é provido gratuitamente o pacote JSDK (Java Servlet Development Kit|) que permite o uso de servlets mesmo em servidores Web cuja característica primária não permitia o uso de servlets (há exemplos de uso com Apache, IIS Microsoft e Netscape Enterprise).

IV.3.1 Implementação de Servlets

De acordo com [31] [34], servlets são uma boa opção para problemas Cliente/Servidor. A característica de robustez e confiabilidade prestadas ao servidor tornam-na uma atrativa solução. Sua implementação é relativamente simples, baseando-se em dois pacotes do JSDK – `javax.servlet` e `javax.servlet.http`.

O processo de execução de um servlet é muito simples e de acordo com [31] pode ser resumido aos seguintes passos:

(a) O servidor cria um novo *servlet*;
(b) O servidor pede ao *servlet* iniciar sua execução (através de um objeto instanciado da classe `ServletConfig`);
(c) O cliente invoca um *servlet* (através de uma requisição em HTML que utilize as *tags* <SERVLET> e </SERVLET>);
(d) Quando o cliente faz uma requisição a um *servlet*, o servidor Web deverá possuir um diretório entitulado /servlets (assim como em cgi,

possuíamos o diretório cg-bin), dessa forma, o servidor estará preparado para receber as requisições;

(e) O *servlet* lê a requisição;

(f) O *servlet* espera um cabeçalho contendo as respostas da operação;

(g) O *servlet* obtém uma cadeia de resposta;

(h) É então, gerada uma página dinâmica contendo as respostas.

IV.4 - JAVA/OBJETOS DISTRIBUÍDOS

A estreita ligação entre Java e browsers Web tornam-na um meio ideal para o desenvolvimento de aplicações baseadas na Web, além de que Java é uma linguagem orientada a objetos de mais fácil utilização que suas predecessoras como o C++.

Uma aplicação para a Web utiliza-se de páginas escritas em HTML para veicular seu conteúdo, no entanto, o uso de *applets* Java pode tornar a interação entre o usuário dessa aplicação e a própria página bem maior. A possibilidade de incluir comandos em páginas HTML permite que determinados tratamentos não necessitem ser executados no servidor HTTP relacionado ao *browser*.

A carga de uma *applet* implica na carga de uma ou várias classes. Classes locais não precisam ser trazidas do servidor, podem ser utilizadas as classes presentes na grande maioria dos *browsers* e interpretadas pela Máquina Virtual Java (MVJ) embutida nos mesmos. De outra forma, classes de origem externa, podem ter o seu código transportado para o *browser* onde se utiliza a aplicação e novamente ser interpretada pela MVJ do *browser*.

Ao passo que nas alternativas anteriores pouco processamento era reservado ao cliente da requisição, o uso de *applets* diminui a carga de trabalho reservada ao servidor. Aplicações CGI, por exemplo, restringiam o tratamento da interface para o lado do cliente. Utilizando-se *applets* o servidor HTTP relacionado ao *browser*, praticamente apenas tem de se importar em tratar as outras operações HTML inclusas na página [23].

No entanto, Java não é suficiente para suprir as necessidades de uma aplicação realmente distribuída. De acordo com [31], não se verifica em Java um real suporte ao paradigma Cliente/Servidor.

Os objetos criados em Java, apesar de possuírem a portabilidade em relação a plataformas, não tem suporte à distribuição e conseqüente integração dos mesmos. A adequação ao paradigma Cliente/Servidor é suprida através de Objetos Distribuídos, pois dessa forma, objetos que antes não podiam se comunicar passam a ter uma infra-estrutura para a integração dos objetos entre si.

Duas tecnologias podem ser utilizadas para efetuar essa integração: *Remote Method Invocation* (RMI) e CORBA. A tecnologia RMI é uma alternativa puramente Java, mas, que não permite a real integração de plataformas. Conforme se verifica na Figura 12, na tecnologia RMI para que se efetive a comunicação entre objetos, faz-se necessário uma adaptação, ou seja, o uso de uma API que permite a comunicação entre objetos distintos.

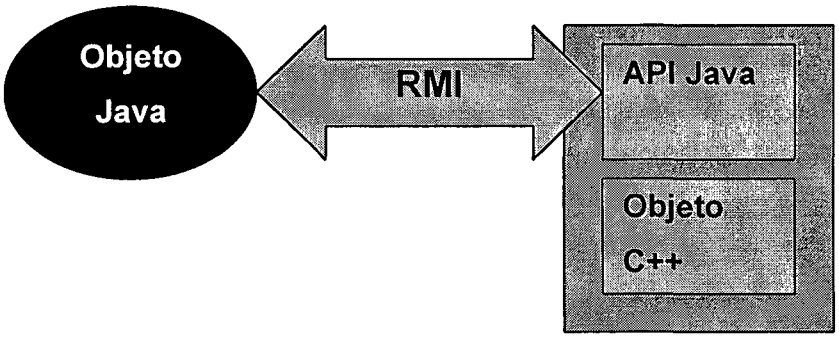


Figura 12 - Invocação de um método em um Objeto Distribuído através de RMI/Java.

Em determinados aspectos trata-se de uma boa solução, haja visto que o uso da API de conversão não seria necessária se os objetos fossem ambos Java. De outra forma, quando se pensa em uma aplicação realmente distribuída, não é suficiente apenas permitir que os objetos comuniquem, é necessário estabelecer uma infra-estrutura, onde: [31]

- aplicações possam encontrar objetos que estejam na rede;

- aplicações consigam identificar propriedades de objetos de forma a requisitar seus serviços;
- aplicações possam se garantir quanto a integridade funcional dos objetos que estão sendo requisitados; e
- aplicações possuam suporte a vários tipos de serviços, intrínsecos a uma aplicação distribuída: controle de concorrência, segurança, nomeação distribuída e recuperação de falhas.

Como já descrito, a arquitetura CORBA é uma real solução para Objetos Distribuídos onde ocorre a integração esperada. Os serviços estabelecidos por CORBA garantem a infra-estrutura preconizada anteriormente, sendo dessa forma uma alternativa Cliente/Servidor Java [22].

Muitos aspectos aproximam Java e CORBA:

- a linguagem de definição de interfaces (IDL) de CORBA possui tipos de dados facilmente mapeados aos tipos de Java;
- os mecanismos de herança de interfaces são idênticos; e
- ambos possuem a noção de interfaces abstratas.

CORBA e Java são tecnologias complementares. Enquanto Java é atualmente estabelecida como uma arquitetura para o desenvolvimento de aplicações para a Web, CORBA provê um poderoso conjunto de ferramentas para o desenvolvimento e a disponibilização de aplicações distribuídas [32]. Dessa complementação, Java e CORBA podem ser propostos como infra-estrutura para a solução de várias categorias de problemas intrinsecamente distribuídos, dada a característica de execução dos mesmos, conforme se verifica na Figura 13.

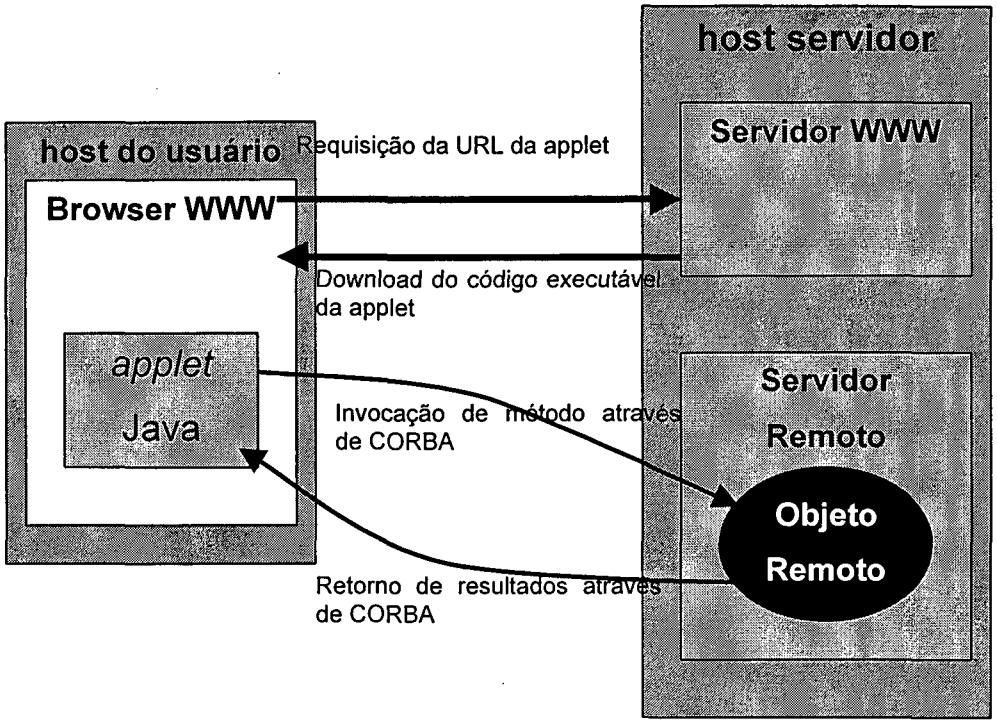


Figura 13 - Visão geral do uso de CORBA e Java [22].

Um problema no qual a associação entre Java e CORBA pode ser útil, é na comunicação entre agentes. Dois exemplos dessa aplicação estão em [15] e [30]. Ambos relatam experiências na construção de agentes para garantia da segurança do acesso a dados na Internet. CORBA e Java se complementam neste aspecto, dado o fato de que Java oferta várias implementações de serviços de segurança. Em específico, a API `java.security`, pode ser utilizado, em conjunto com as facilidades CORBA para a obtenção de serviços de segurança para aplicações distribuídas.

De outra forma, a opção CORBA/Java também é útil no estabelecimento de uma infra-estrutura para a disponibilização de serviços na Web que necessitem acessar Bancos de Dados. Duas soluções são relatadas em [13] e [14], no entanto, trata-se de um *framework* para a manipulação de Bancos de Dados Federados, ou seja, uma solução para consulta e modificação de dados em uma estrutura heterogênea de Bancos de Dados. Conforme será descrito na próxima seção, o modelo proposto por este trabalho utiliza de Java com suas

APIs de segurança e de acesso a Bancos de Dados (JDBC) para dispor uma aplicação distribuída na Web.

IV.5 - AMBIENTE JAVA/CORBA PARA O ACESSO A BDS NA INTERNET

A questão associada a este tópico é: como através de CORBA e Java pode-se fazer um acesso a um Banco de Dados ? Em específico, como um usuário de um *browser* pode através de sua conexão à rede Internet, acessar um Banco de Dados ?

Como este trabalho baseia-se na WWW como ambiente básico, *applets* Java são utilizadas como interface com o usuário. Neste caso, para prover um acesso a Bancos de Dados, protocolos de conversão devem ser utilizados a fim de que quaisquer requisições enviadas ao usuário sejam interpretadas. A conversão, neste caso, é feita pela *Application Program Interface* (API) Java JDBC (*Java Database Connectivity*) que permite que requisições sejam convertidas em linguagem SQL, interpretadas pelo SGBD em questão e após isso retornadas ao usuário. A Figura 14 apresenta uma caracterização destas operações.

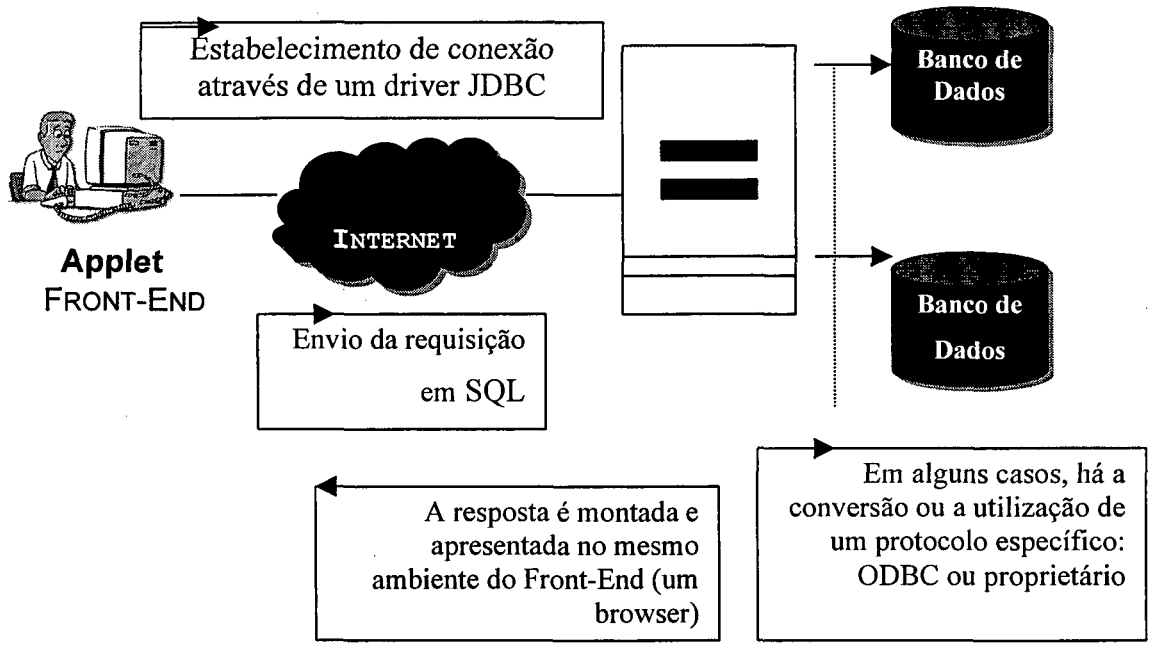


Figura 14 - Requisições a BDs remotos através da API JDBC.

Com o *Java Database Connectivity* [40] é possível utilizar-se de funções, componentes de sua biblioteca, para acessar Bancos de Dados, através da linguagem Java. Como os Bancos de Dados possuem padrões distintos, cada fornecedor oferece um *driver* de conexão para seu produto, no entanto, todos baseiam-se no uso da *Structured Query Language* (SQL) para tratar requisições. Essa API, denominada JDBC é, na verdade, um recurso utilizado para se permitir o uso de SQL através da linguagem Java.

A classe *DriverManager* é uma componente dessa API e é utilizada para abrir uma conexão com um BD através de um *driver* JDBC, que precisa estar registrado antes de realizar uma conexão. Quando uma conexão é iniciada, o *DriverManager* escolhe um *driver* de uma lista de *drivers* disponíveis para efetuar a conexão, dependendo da URL especificada. Uma vez estabelecida a conexão com sucesso, as chamadas para consultas e buscas de resultados serão feitas diretamente no *driver* JDBC. A Figura 15 ilustra como são feitas essas operações.

```
import java.lang.*;
import java.awt.*;
import java.sql.*;
import java.applet.*;

public class Exemplo extends Frame {
static {
            try {
            Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
            } catch (ClassNotFoundException e) {
             e.printStackTrace();}
           }

public boolean pesqDados() {
try {
      Connection con = DriverManager.getConnection("jdbc:odbc:spc");
      Statement stmt = con.createStatement();
      ResultSet rs = stmt.executeQuery("SELECT * FROM Clientes ");
      while (rs.next()) {
              String Name = rs.getString(2);
              nomeTF.setText (Name); }
      rs.close();
      stmt.close();
      con.close();
    } catch (Exception e) {
        System.out.println ("Erro fatal");}
      return true;
}
```

Figura 15 - Uso da API JDBC para acesso a um BD.

Para que Java seja utilizado como *front-end* para o usuário, ou seja, interface de entrada, *browsers* Web devem ser utilizados para complementar essa interface. No entanto, algumas considerações relacionadas à postura de segurança padrão de Java em todo os *browsers* devem ser observadas [01] [04] [10]:

- Modelo *Sandbox,* no qual a arquitetura de segurança Java se baseia, estabelece que *applets* apenas podem estabelecer conexões de rede com o seu *host* de origem; e
- que *applets* apenas podem aceitar conexões de rede de seu *host* de origem.

Ou seja, quaisquer interações via rede através de *applets* somente ocorrem com seus *hosts* de origem. Em se tratando de um ambiente baseado em Objetos Distribuídos, *gateways* podem ser utilizados para a flexibilização desta política de segurança adotada. Em específico, o ORB *Visibroker*, comercializado pela *Visigenic* apresenta um serviço adicional chamado *Visibroker Gatekeeper* [12]. O *Visibroker Gatekeeper* facilita aplicações distribuídas a comunicarem com outros objetos servidores mesmo através de um *firewall* sem com isso comprometer a segurança da rede.

Conforme será discutido no próximo capítulo, as flexibilidades oferecidas pela nova versão do kit de desenvolvimento Java JDK 1.2, permitem que permissões específicas sejam configuradas [38] [60]. Desta forma, operações antes impedidas podem ser executadas, por exemplo, permitir que uma *applet* execute a operação de escrita em um arquivo ou Banco de Dados.

Para ressaltar a possibilidade do Modelo *Sandbox* ser flexibilizado, dois autores citam outros princípios que, inclusive, facilitam a comunicação entre ORBs através de applets:

> ***"As restrições do Modelo Sandbox podem ser amenizadas através do uso de código assinado digitalmente"*** [30] e ainda: "... ***podem ser feitas conexões a quaisquer servidores e também receber mensagens***" [12].

Enfim, deve ser considerado o tratamento da segurança dada à comunicação ORB-Cliente e entre ORBs. Neste caso, a solução adotada pelo produto *Visibroker for Java* 3.3 é um pacote adicional, baseado no protocolo SSL (*Secure Socket Layer*), chamado *Visibroker SSL Pack*, cuja ativação se dá pelo próprio *Gatekeeper* mencionado.

A função deste pacote é habilitar chamadas seguras entre uma *applet* e um objeto servidor. Ou seja, entre o *applet* e o ORB e entre o ORB e o *Gatekeeper* todas as comunicações serão garantidamente seguras, tendo seu conteúdo criptografado e cliente e receptor autenticados.

O protocolo SSL é definido para comunicações privadas e autenticadas. Seu escopo de atuação baseia-se nas camadas de transporte confiáveis (tais como TCP), mas, estando abaixo de protocolos de aplicação (tais como HTTP) ou ainda abaixo de APIs (tais como as de Java). O SSL permite que agentes possam escolher dentre um conjunto de algoritmos de criptografia e de esquemas de autenticação para ser executado. A API `java.Security`, componente do *Java Developer Kit* (JDK) apresenta a possibilidade de serem utilizados dois algoritmos: RSA (Rivest-Shamir-Adleman) e DES (Data Encryption Standard), o que nos permite tratar as questões de segurança do ambiente através de Java.

Portanto, os seguintes recursos computacionais são providos para o fornecimento de um ambiente onde Bancos de Dados sejam utilizados em *sites* na Web, cujas transações sejam seguras e cuja distribuição seja transparente para o usuário, conforme [39]:

Java, utilizada como:

- *front-end*, ou seja, como interface para a requisição de operações e a recepção de respostas;
- provedora de serviços de segurança, através de sua API *java.Security*, tais como: verificação da identidade de usuários (através de controle de acesso baseado em autenticação), verificação da "visita" de agentes (através de

auditoria) e fortificação de ambiente local contra ataques (através de criptografia);
- intermediária no acesso a Bancos de Dados remotos, através de sua API JDBC.

CORBA, utilizada através:
- do ORB comercial Visibroker, fornecendo serviços de tratamento dos objetos que estarão distribuídos.

SGBDs, utilizados como provedores de operações vitais a um SBD, tais como:
- armazenamento dos dados; e
- processamento das requisições.

IV.6 - Características do Ambiente Proposto

A estruturação do ambiente onde Java e CORBA sejam utilizados conforme as considerações da seção anterior, está descrito na tabela seguinte.

Fase	Descrição
(1) Ambiente SBDD	estabelecimento de um ambiente Web no qual possam ser feitos acessos a Bancos de Dados remotos através de *applets* e da utilização de um ORB comercial.
(2) SBDD_Web seguro	através de Java, implementar: controle de acesso e autenticação de usuários.
(3) Auditoria	construção de um agente que envie relatórios de alarmes, baseado em tentativas de acesso indevidas, tais como: *logins* insistentes e mal sucedidos e tentativa de acessos sucessivos a recursos proibidos.

Tabela 4 - Fases para a consecução do ambiente seguro de acesso a BDs na Web.

Na Fase 1, identificada como Ambiente SBBD, o uso da API JDBC provê a possibilidade de dispor na Web operações a serem executadas pelos usuários, no entanto, um outro nível de usuário deve ser contemplado: o Administrador do Banco de Dados (DBA).

A administração dos dados, conforme [41], é uma intrínseca tarefa de monitoração. Decisões importantes baseadas nas visões que usuários ou grupos de usuários devem ter do Banco de Dados são identificados pelo DBA.

Desta forma, na Fase 1, também são contempladas operações que ajudem o DBA na organização das visões dos usuários. Em particular, será provido um Banco de Dados Seguro cujas informações podem ajudar o sistema tomar decisões tais como:

- serviços permitidos;
- tipo de comunicação exigida na operação;
- restrição de *hosts*; e
- verificação de assinatura digital.

Para que o DBA possa executar essas operações é provido um Banco de Dados de informações do usuário, de forma que possam ser estabelecidas as restrições de acesso.

Dessa forma, a Fase 2, denominada *SBDD_Web* seguro, utiliza das informações providas pelo DBA para permitir que as interações sejam seguras. Como já descrito, Java será utilizada através da APi `java.Security` com a finalidade de gerenciar a criação e a verificação de chaves e assinaturas digitais. No entanto, a fim de evitar que comunicações sejam feitas diretamente com o objeto *Security Manager*, um objeto intermediário designado como **procurador** será provido com as seguintes finalidades:

- reportar ao *Security Manager* tentativas de acesso indevidas;
- comunicar ao *Security Manager* da intenção de um usuário de estabelecer uma conexão ao BD; e
- monitorar conexões abertas e enviar relatórios de tempo de uso e recursos acessados ao *Security Manager*.

Na Fase 3, o objeto até então denominado *Security Manager* é discutido e implementado. Três funções serão imprescindíveis para a garantia de um ambiente seguro:

- autenticação da comunicação do cliente final com os recursos do BD;
- sumarizar *logs* de acesso, enviados pelo **procurador** a respeito do comportamento das requisições e que estarão disponíveis ao DBA; e

- ter à disposição, por intermédio do DBA, um Banco de Dados específico e seguro, cujos dados forneçam informação suficiente para o *Security Manager* impedir ou não o acesso a determinados recursos.

Para concluir a descrição desejada, a Figura 16 esboça como são incluídos, os objetos propostos na solução:

- procurador
- *Security Manager*
- DBA e o seu módulo de configuração.

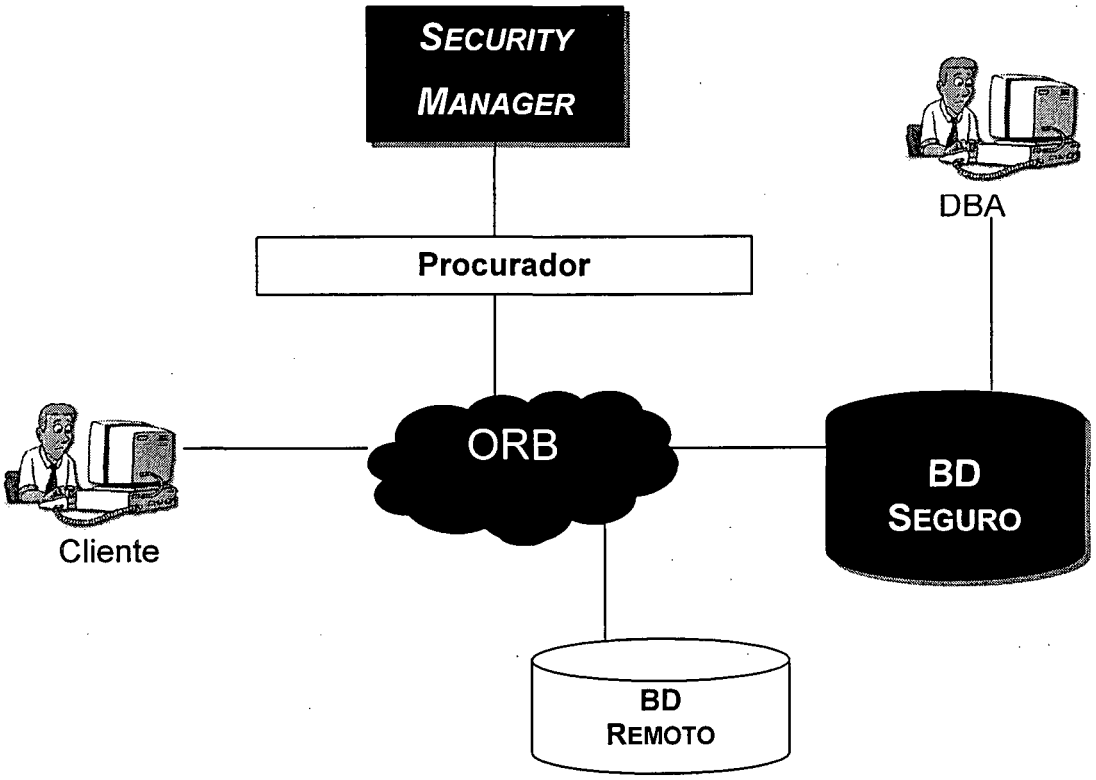


Figura 16 - Visão do ambiente Java/CORBA para acesso a BDs na Web.

V- EXTRATO DE CONTA TELEFÔNICA NA WEB

Para que um serviço de Extrato de Conta Telefônica na Web fosse concebido, algumas empresas que disponibilizam serviços na Web e que se preocupam com segurança foram consultadas. Alguns exemplos podem ser fornecidos:

***Personal Home Banking* do Banco do Brasil.**

Este serviço do Banco do Brasil [44] permite a execução de operações perigosas para os clientes deste banco. Desta forma, um preparo muito grande para que o cliente incorpore a filosofia de segurança é feito através de dicas e manuais que explicam situações que podem ser vulneráveis. O uso de Java se verifica na implementação dos serviços de segurança, além da aplicação do protocolo SSL como fonte para o controle de acesso e a integridade dos dados. Além disso, este *site* preocupa-se, também que o cliente esteja acessando o serviço do local correto, fornecendo ao usuário a possibilidade de identificar a certificação digital do mesmo.

Serviços Visa

A administradora de cartões de créditos e instituição financeira Visa [45] utiliza das informações armazenadas em *cookies* para monitorar as conexões feitas às suas páginas. Assim como o serviço de *home banking* do Banco do Brasil, possui forte divulgação da importância da segurança no comércio praticado na Internet, não só através de informações veiculadas nas páginas, mas, também através da divulgação do protocolo SET. Uma característica negativa é a não certificação digital do site, o que pode permitir a incidência de ataques do tipo *spoofing*, ou seja, que ludibriam o usuário fazendo-o imaginar que uma determinada página "pirata" é real.

Amazon Books

Outra empresa [46] que confia no protocolo SSL para a transação de suas operações. Neste *site*, percebe-se que há a possibilidade de criar

situações alternativas para a garantia da segurança. Por exemplo, para o usuário que não deseja informar o número completo de seu cartão de crédito, a empresa permite o fornecimento dos últimos números e da data de vencimento através de fax. Dessa forma, outras consultas necessárias não seriam feitas pela rede, mas, através de telefone.

As características observadas nos *sites* pesquisados contribuíram para a concepção do serviço de Extrato Telefônico via Web. Características como a certificação digital, o uso do protocolo SSL e a divulgação de noções de segurança para o usuário são utilizados na implementação.

Este estudo de caso está inserido, no momento, em um conjunto de outros projetos que estão em desenvolvimento no Laboratório de Redes e Gerência (LRG) da Universidade Federal de Santa Catarina (UFSC).

O serviço de fornecimento de Extrato de Conta Telefônica na Web possui, então as seguintes características, no tocante à segurança:

- por ser um serviço pago, o usuário precisa se cadastrar para obter o serviço;
- o fornecimento das informações se dá mediante uma senha. Esta senha é fornecida pelo próprio usuário, no entanto, a própria empresa se encarregaria de criar as chaves privada e pública para que a identidade do mesmo pudesse ser autenticada;
- a própria página da empresa necessita de uma forma de certificar ao usuário sua identidade. Isso pode ser feito por intermédio de um certificado digital; e
- para que o extrato da conta telefônica do usuário não seja interceptada, são usados mecanismos de segurança, baseados em criptografia.

Outras características estão relacionadas à implementação também:

(a) como se trata de um serviço intrinsecamente distribuído, haja visto que pode atender a várias centrais da mesma empresa telefônica, é necessário permitir que sejam distribuídos objetos que possam atender a usuários de

todas as localidades atendidas pela empresa e que consequentemente possuem suas informações em *hosts* distintos;

(b) a conta telefônica para ser emitida precisa ser contabilizada; e

(c) a aplicação precisa ter dispositivos que permitam-na recuperar-se de falhas.

As características (b) e (c) são contempladas em outros projetos.

Para o desenvolvimento, foram utilizados as seguintes ferramentas:

Tabela 5 - Produtos utilizados na validação do estudo de caso.

Visigenic Visibroker for Java 3.3	ORB CORBA, cuja tradução de IDLs é específica para a linguagem Java
Java Developer Kit 1.2	Ferramenta de desenvolvimento de aplicações na linguagem Java, cujas características serão melhor discutidos na próxima seção.
HTML Converter for Java	Conversor de páginas escritas HTML para a inclusão de *tags* que identifiquem a presença de uma *applet* escrita na versão 1.2 do JDK.
policytool	Ferramenta do kit JDK 1.2, cuja função é permitir que sejam estabelecidas permissões de acesso e modificação de componentes através de *applets*.
keytool	Ferramenta do kit JDK .1.2, cuja função é gerar chaves criptográficas para os usuários.

V.1 - CARACTERÍSTICAS DAS FERRAMENTAS UTILIZADAS

As ferramentas utilizadas no desenvolvimento deste estudo de caso apresentam algumas peculiaridades que são discutidas neste item. Estas características permitem que sejam identificadas soluções para o tratamento da segurança no ambiente proposto.

V.1.1 - VISIGENIC VISIBROKER 3.3 FOR JAVA

Esta ferramenta foi escolhida para a produção dos exemplos dada as facilidades para se obter uma versão gratuita para uso em tempo limitado e por ter um serviço de atendimento bom. No entanto, outras ferramentas poderiam ter sido escolhidas.

As características chave apresentadas por essa ferramenta são:

(a) **Interface de Repositórios** - de onde os objetos podem identificar a presença de serviços de outros objetos no ORB;
(b) **DII** - programas cliente podem obter informação sobre a interface do objeto e, dinamicamente construir requisições para se comunicar com o mesmo;
(c) **Smart Binding** - é um mecanismo que provê a escolha do melhor mecanismo de transporte sempre que um cliente se conecta a um objeto servidor;
(d) **Smart Agents** - torna fácil a tarefa de se identificar a presença de objetos no ORB. De outra forma, permite que um cliente automaticamente reconecte-se a um objeto servidor se ocorrer alguma falha;
(e) **Location Service** - é uma extensão da especificação CORBA que provê facilidades gerais para a localização de instâncias de objetos;
(f) **Compilador idl2java** - compilador que transforma especificações em IDL para interfaces em Java. Nesta versão, este compilador possui o seu próprio pré-processador (*vbjc* e *vbj*);
(g) **Communication event handlers** - este mecanismo notifica clientes e implementações de objetos de eventos do sistema. Essa característica é útil na implementação de serviços de *debbuging* e segurança;
(h) **Gatekeeper** - conforme já discutido, trata-se de um módulo que permite a clientes executarem chamadas a objetos que não estejam em seu módulo servidor;
(i) **SSL Pack** - ferramenta que permite a implementação de serviços de segurança.

Nem todas as características se encontram disponíveis na versão pesquisada. Desta forma, alguns recursos são implementados em Java, como por exemplo, os serviços de segurança, encontrados na API Java `java.security`.

V.1.2 - JAVA DEVELOPER KIT 1.2

Melhorias foram incluídas na atual proposta de segurança do JDK 1.2, no entanto, mantendo-se os mesmos níveis de verificação da arquitetura de segurança. As extensões verificadas são:

- possibilidade de administrar e configurar políticas de segurança;
- possibilidade de efetuar controle de acesso;
- novos serviços de criptografia; e
- gerenciamento de chaves e certificados.

O conceito fundamental da nova proposta é o **domínio de proteção**, um mecanismo conveniente para o agrupamento e isolamento de unidades do sistema que necessitam ser protegidas. Dessa forma, a flexibilidade que vem sendo mencionada é verificada na adoção de políticas de segurança que implementam permissões a *sites* e usuários no acesso a recursos do sistema.

De acordo com Gong [06], domínios de proteção também podem garantir o controle do acesso a recursos críticos. Através dos pacotes `java.security.Policy` e `java.security.Permission`, pode-se implementar controle de acesso a recursos tais como acesso a arquivos. Na Figura 17, podem ser verificados dois trechos de código onde essa implementação é feita:

```
grant CodeBase "http://origem.com.br/joao",
Signed by "*" {

permission java.io.FilePermission "read, write", "tmp/*";
permission java.net.SocketPermission "connect", "*.origem.com.br";
};

grantCodeBase "/home/joao/artigos"
Signe by "self" {

permission java.io.FilePermission "read, write, delete",
"/home/joao/-";
};
```

Figura 17 - Permissão de acesso a arquivos e estabelecimento de conexão de rede.

Do exemplo acima, verifica-se que:

- classes específicas relacionadas à permissão de acesso devem ser utilizadas `java.io.FilePermission` e `java.net.SocketPermission`);
- a assinatura digital é vital para a concessão da permissão, mesmo que ela não seja necessária (como em `Signed by "*"`);
- a inflexibilidade do Modelo *Sandbox* não impede a execução de operações delicadas, tais como o acesso para leitura e escrita de arquivos remota, no entanto, sem a verificação de uma assinatura, o mesmo fica impedido.

A gerência das permissões é feita pela ferramenta *policytool*, ou através da configuração manual do arquivo `java.policy` localizado no diretório `\jdk1.2\jre\lib\security`. Neste trabalho, a mesma foi utilizada com o intuito de permitir que as informações do usuário fossem armazenadas em um BD localizado no *host* servidor, assim como permitir que o usuário, implicitamente, consultasse informações de sua chave privada. A Figura 18 exibe as permissões que foram necessárias para a aplicação.

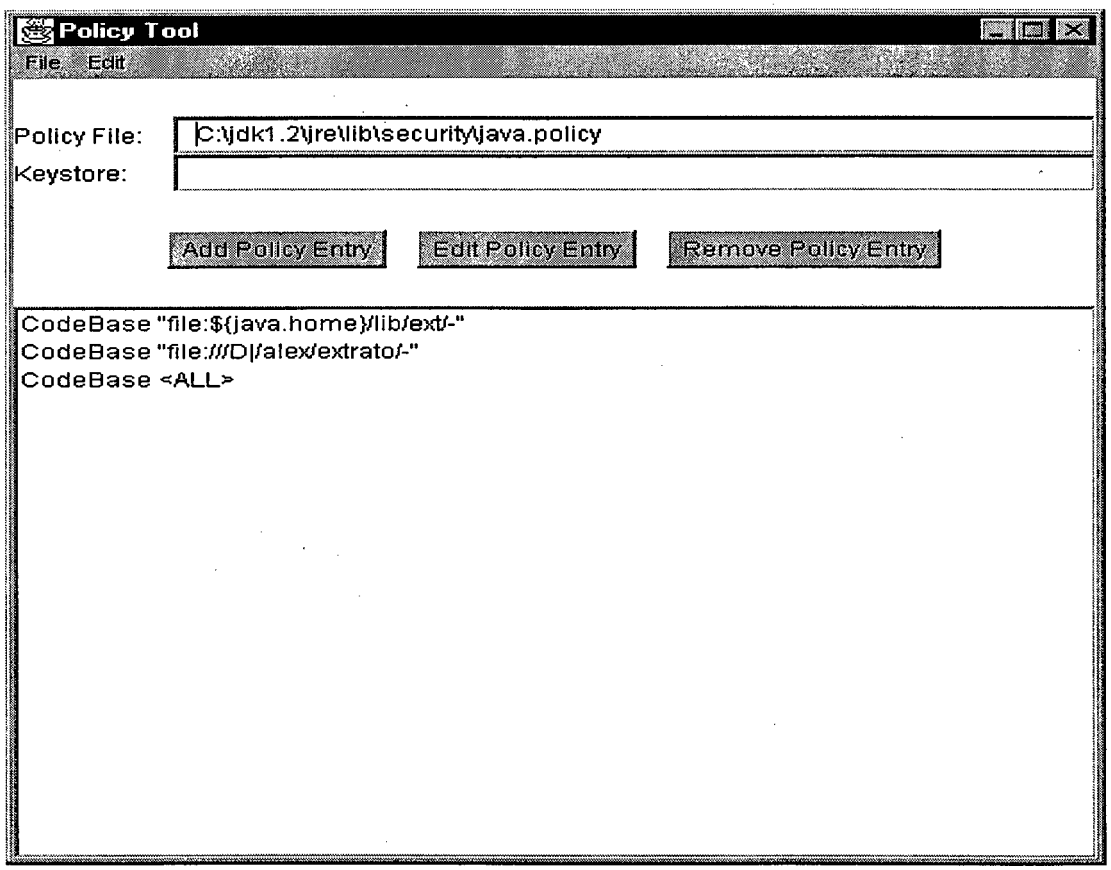


Figura 18 - Permissões concedidas a *applets*.

Para a execução da aplicação, uma restrição verificada nos *browsers* necessita ser retificada. Os *browsers* ainda não possuem todas as classes utilizadas pelo JDK 1.2 reconhecidas pelas MVJs inclusas nos mesmos. Dessa forma, um *plug-in* é necessário para que possam ser reconhecidas tais classes. Na verdade, a versão do JDK 1.2 para a plataforma Windows já inclui a instalação de um ambiente de execução Java (JRE - *Java Runtime Environment*) compatível, no entanto, as páginas precisam receber *tags* complementares para que um tratamento diferenciado seja dado. A Figura 19 ilustra o Java *HTMLConverter*, sua função é incluir informações adicionais nas páginas HTML que possuem chamadas a classes produzidas no JDK 1.2.

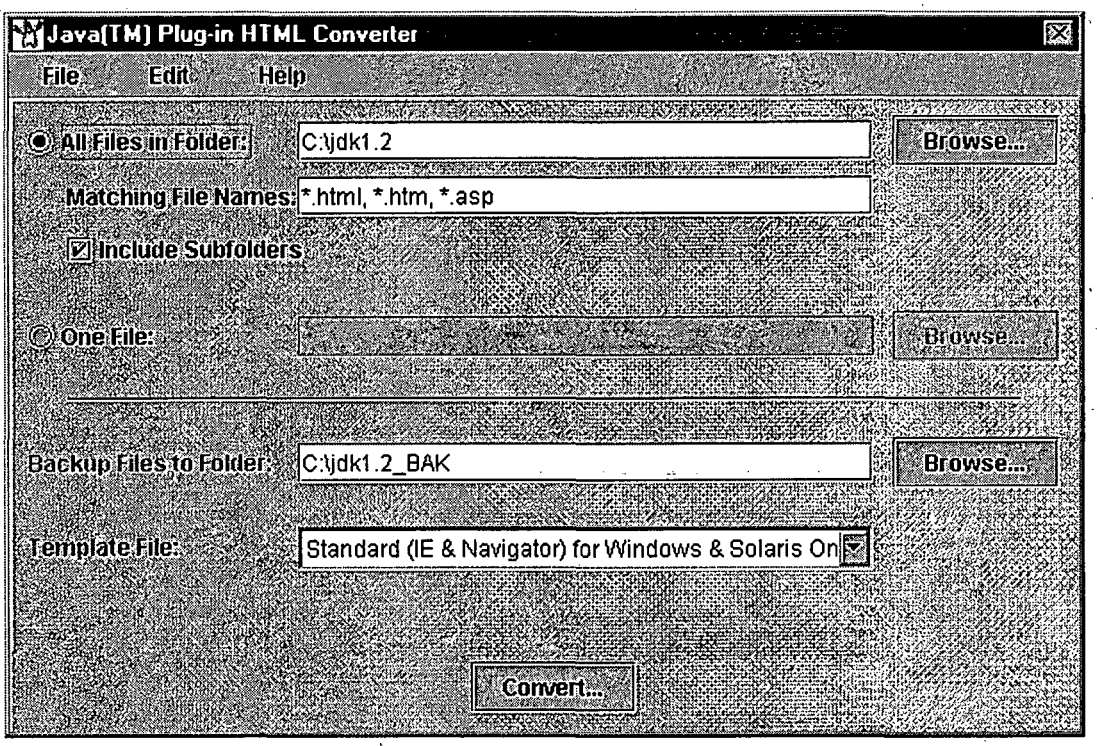


Figura 19 - Ferramenta HTMLConverter.

As características originais das páginas não são descartadas. Conforme se verifica na Figura 19, arquivos de recuperação ficam armazenados em `C:\jdk1.2_BAK`, sendo substituídos pelos novos arquivos.

Prevê-se que as novas versões dos browsers incluam o reconhecimento deste tipo de *script*, no entanto, para que os novos recursos verificados pudessem ser utilizados, foi necessária a conversão, conforme se verifica na Figura 20.

```
<!--"CONVERTED_APPLET"--><!-- CONVERTER VERSION 1.0 --><object
classid="clsid:8AD9C840-044E-11D1-B3E9-00805F499D93"
WIDTH = 400 HEIGHT = 300
codebase="http://java.sun.com/products/pl<object
classid="clsid:8AD9C840-044E-11D1-B3E9-0<param NAME = CODE VALUE =
"Client.class" ><param NAME="type" VALUE="application/x-java-
applet;version=1.2"><COMMENT><embed type="application/x-java-
applet;version=1.2" java_CODE = "Client.class" WIDTH = 400 HEIGHT =
300   pluginspage="http://ja<embed type="application/x-java-
applet;version=1.2" <noembed></COMMENT></noembed></embed></object><!--
<APPLET  CODE = "Client.class" WIDTH = 400 HEIGHT = 300 >


</APPLET>
--><!--"END_CONVERTED_APPLET"--></td>
```

Figura 20 - Trecho de uma página HTML convertida pelo HTMLConverter.

V.2 - IMPLEMENTAÇÃO

A implementação inicia-se pelo módulo responsável por incluir novos usuários e consequentemente gerar chaves criptográficas para que o mesmo possa cadastrar sua senha de acesso.

Foi adotado que ao utilizar a página de acesso aos serviços, o usuário fornece suas informações pessoais. Após ter-se cadastrado, o mesmo recebe em casa uma aplicação que instala em sua máquina informações sobre o *site* do serviço e a própria chave privada do mesmo. Neste instante é que o usuário fornece sua senha, pois, dessa forma sua chave privada que é utilizada para criptografar a senha, não pode ser interceptada pela rede, por ter sido transportada por correio.

Um usuário poderia argumentar como poderia confiar na veracidade da chave criada, assim como na identidade do *site* de serviços. A chave privada criada somente terá utilidade no site que o criou, de outra forma, pode ser utilizada certificação digital para o usuário poder confiar em quem está acessando.

A Figura 21 esboça o instante em que o usuário fornece seus dados pessoais. Sua inclusão no cadastro automaticamente dispara a criação das chaves criptográficas.

Figura 21 - Cadastro de usuário no Sistema de Extrato Telefônico na Web.

Verifica-se, na carga da *applet* um ligeiro declínio da performance, haja visto a necessidade da carga do *plug-in* que permite a interpretação da classe produzida no JDK 1.2. No entanto, conforme já discutido na seção IV.5, este fato não impede de considerar o uso de Java/CORBA a solução mais indicada, pois: trata-se de um serviço intrinsecamente distribuído e o tempo de carga da *applet* apenas compromete no primeiro instante, após a primeira utilização da *applet*, a mesma já está carregada na pilha de execução da MVJ. Por outro lado, na alternativa CGI/HTML, por exemplo, a cada nova requisição um novo acesso a um programa CGI é necessário.

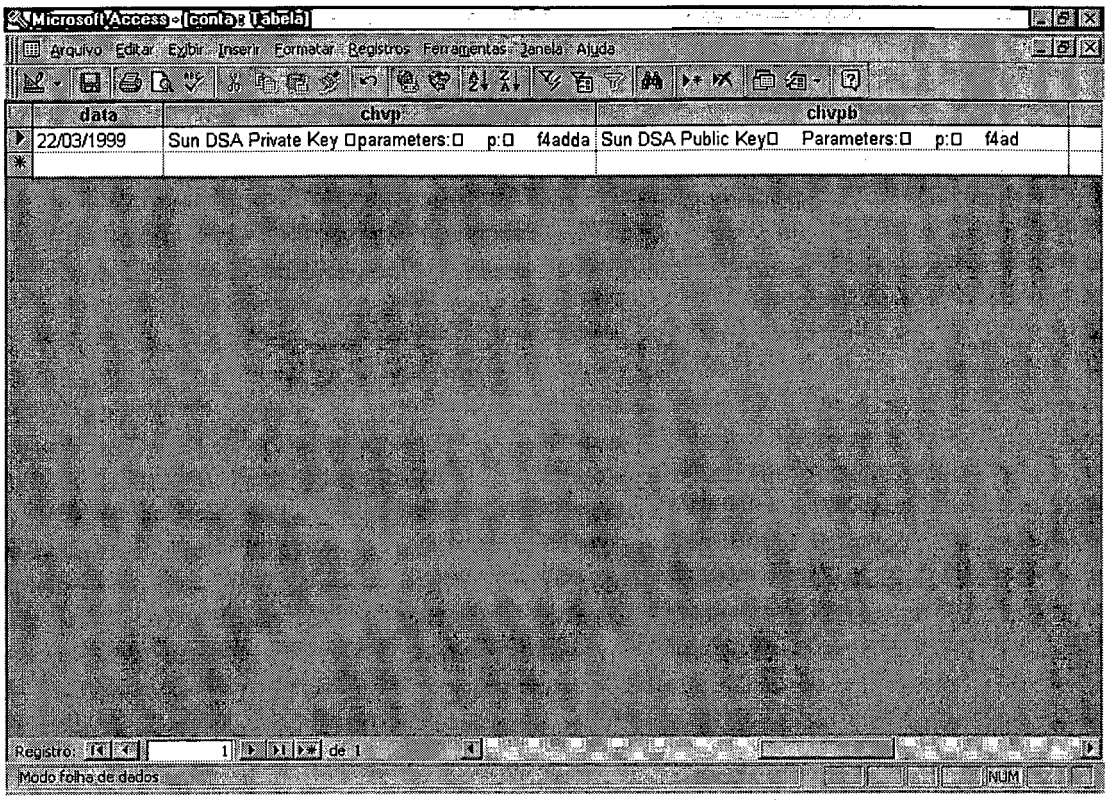


Figura 22 - Chaves Privada e Pública geradas.

As chaves criadas são armazenadas em um Banco de Dados, conforme verifica-se na Figura 22. Este procedimento facilita a tarefa de modificação de senha do usuário, assim como permite o fornecimento imediato da instalação da chave privada do usuário quando o mesmo perdê-la, ou até mesmo quiser recadastrar-se.

Para a implementação dos serviços de segurança foram utilizadas classes da linguagem Java pertencentes à API `java.security`. Em específico foram utilizadas as seguintes classes:

- `java.security.KeyPairGenerator`: cuja função é contactar um provedor de serviços de criptografia (CSP) para que o mesmo dispare o processo de criação de chaves;
- `java.security.PrivateKey`: responsável pela criação de chaves privadas;
- `java.security.PublicKey`: responsável pela criação de chaves públicas;

- `java.security.Signature`: sua função é contactar um provedor de serviços de criptografia (CSP) para que o mesmo dispare o processo de criação de assinatura digitais;
- `java.security.KeyFactory`: utilizada para instanciar uma chave pública produzida com o algoritmo DSA (Digital Signature Algorithm);
- `java.security.spec.X509EncodedKeySpec`: chaves podem ser instanciadas a partir de um arquivo externo, caso o usuário já possua sua chave privada. Desta forma, é verificada sua conformidade com o padrão X.509.

Alguns procedimentos complementam a política de segurança. A fim de que a autenticidade do *site* seja constatada, é necessário que o usuário confie que está utilizando o serviço do local correto. Pode-se, então prover um certificado digital para o site, através de autoridades de certificação que disponibilizam este serviço na Internet. Na Figura 23, tem-se o site da autoridade certificadora Verisign.

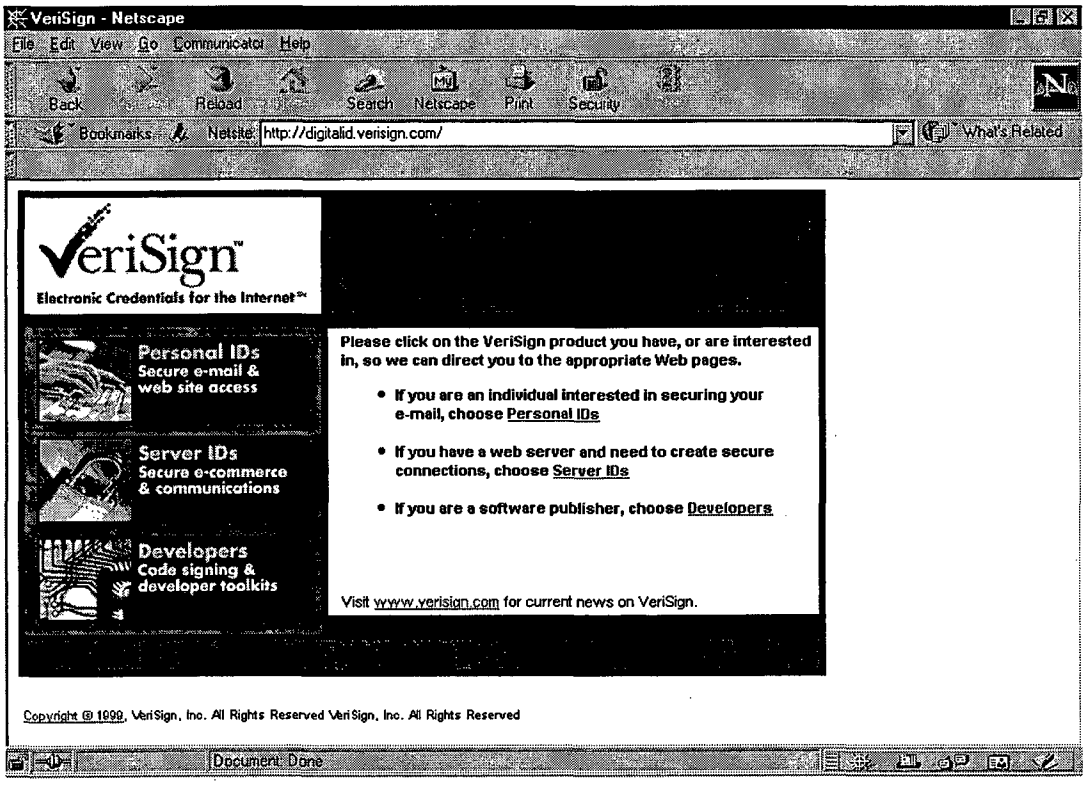


Figura 23 - Autoridade Certificadora Verisign.

Através de um certificado digital a empresa pode certificar seus usuários de que está sendo provido o serviço e não um outro *site*. De qualquer forma, deve-se lembrar que sempre é extremamente aconselhável ao usuário utilizar as mais novas versões dos browsers, pois, os mesmos possuem as mais atuais implementações dos mecanismos de segurança.

O *Netscape Communicator*, por exemplo, permite que o usuário perceba a presença de um *site* certificado, ao emitir uma mensagem de aviso para o mesmo. Na Figura 24, percebe-se, também que, as atuais versões destes *browsers*, permitem uma configuração mais detalhada das características de segurança. No entanto, são configurações destinadas ao usuário, mas que facilitam a adoção de uma política de segurança mais rígida, o que seria, inclusive uma vantagem a mais para o usuário.

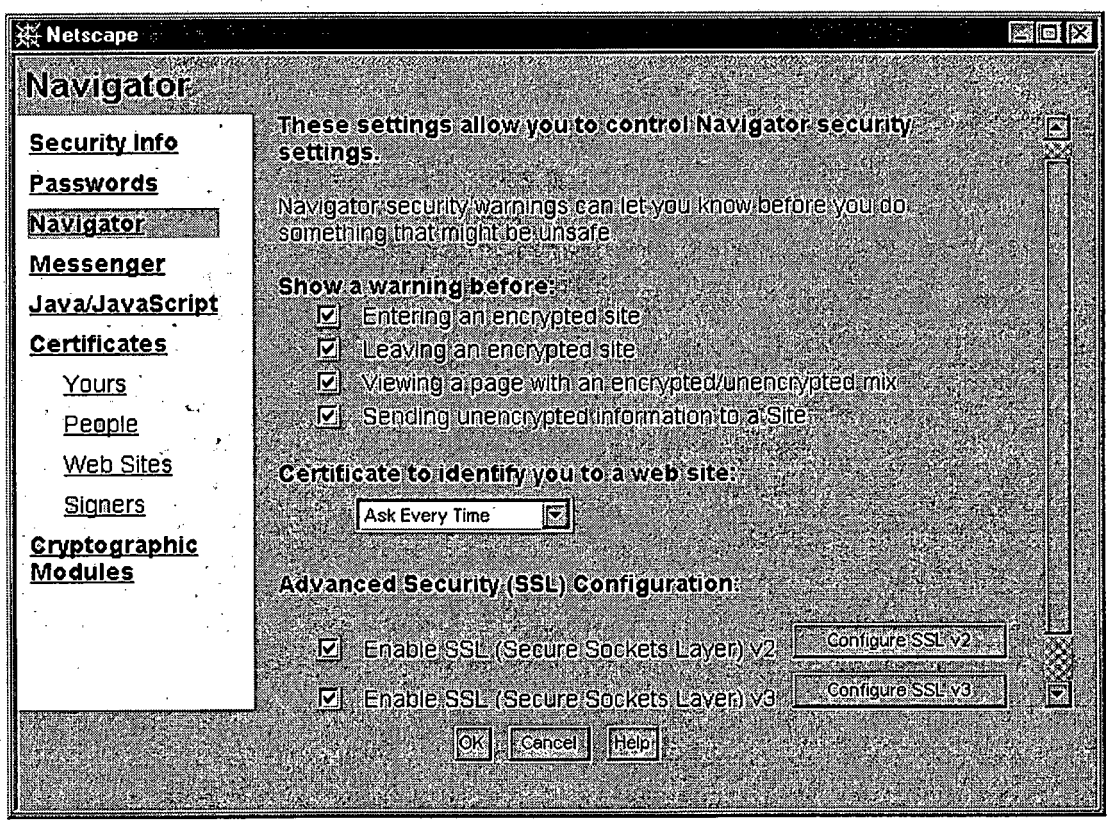


Figura 24 - Configuração da segurança no *Netscape Communicator*.

No Anexo A encontram-se trechos de código utilizados na implementação, assim, como explicações sobre a configuração da segurança nos *browsers* utilizada.

VI - Conclusões e Perspectivas Futuras

Todas as alternativas estudadas apresentam características positivas para o estabelecimento de associações seguras na consulta a Bancos de Dados através da Web.

O importante é prover ao ambiente ferramentas e tecnologias suficientes. Como a cifragem é primordial para a garantia do tríplice requisito de segurança: Confidencialidade, Integridade e Autenticação, protocolos devem ser utilizados. A grande maioria das ferramentas pesquisadas possuem suporte para SSL, no entanto, o suporte para o gerenciamento para chaves públicas necessita da intermediação de alguma entidade certificadora. Esse fato dificulta o processo de validação, pois, entidades certificadoras exigem o pagamento para que possam emitir os certificados digitais. Além desse fato, percebeu-se a importância do uso de *firewalls* para colaboração em aspectos que não foram amplamente abordados neste trabalho: a detecção de intrusão.

O desenvolvimento do protótipo indicado foi complexo. No entanto, o objetivo foi alcançado. É verificado que Java/CORBA é uma boa solução para o estabelecimento de consultas a Bancos de Dados na Web e que serviços de segurança importantes podem ser implementados.

O ambiente implementado, permite, então, que seja disponibilizado um *site*, onde o mesmo executa serviços de consulta a Bancos de Dados e o usuário possui a garantia de que está ocorrendo o controle de acesso e a autenticação dos parceiros na comunicação. Para tanto, além de Java e CORBA, fez-se necessário explorar as facilidades oferecidas pelos *browsers*.

Outras conclusões deste trabalho são reportados na seguinte estrutura:

- dificuldades encontradas;
- resultados esperados e obtidos; e
- trabalhos futuros.

VI.1 - DIFICULDADES ENCONTRADAS

Uma dificuldade inicial foi a grande quantidade de conhecimento teórico exigido. Várias tecnologias foram estudadas e, em alguns casos, com dificuldade para se obter material de referência específico. Dois exemplos podem ser citados: a arquitetura de segurança do *Internet Explorer* e a tecnologia ISAPI.

Na fase de pesquisa, pouco se encontrou sobre ambientes baseados no kit de desenvolvimento JDK 1.2. Pouco se sabe ainda sobre a implementação de protótipos usando o mesmo, apesar de já terem sido encontradas falhas na arquitetura de segurança proposta.

O fato dos atuais *browsers* ainda não incorporarem o reconhecimento de classes e padrões instituídos no novo pacote de desenvolvimento Java (JDK 1.2), foi uma complicação. No entanto, o uso da ferramenta HTMLConverter em conjunto com o fato de que toda instalação do JDK 1.2 para a plataforma Windows, inclui também a instalação do ambiente JRE 1.2, solucionou esta restrição.

Na fase de implementação, outros problemas foram percebidos. A configuração do ambiente para que o kit JDK 1.2 pudesse ser utilizado é bastante sutil, infelizmente, em alguns instantes a documentação do produto foi ineficaz, o que provocou perda de tempo na pesquisa de soluções intermediárias.

Ainda na fase de implementação, restrições de documentação e de disponibilidade de uma ferramenta, impediram que o ambiente também fosse desenvolvido na plataforma Unix. Dessa feita, o protótipo está construído no ambiente Windows 95/Windows NT.

Outra dificuldade na implementação, são os curtos prazos fornecidos pelas empresas para a fase de experiência do software. Por duas vezes, o software *Visigenic Visibroker* 3.3 necessitou ser reinstalado devido ao término de seu prazo de utilização.

Por fim, para que a finalização deste projeto pudesse ocorrer em tempo hábil, a abrangência das implementações foi restringida. Em conseqüência deste fato, não foi implementado um agente de auditoria para verificar comportamentos perigosos no acesso.

VI.2 - RESULTADOS ESPERADOS E OBTIDOS

Acessar Bancos de Dados através da Web é uma tarefa que exige um nível a mais. Além da prestação de serviços habituais, através de um servidor Web, o mesmo é responsável por administrar conexões a um Banco de Dados, ou seja, o servidor, ou a máquina que presta serviços de aplicações distribuídas devem ser fortemente protegidos.

Especificamente, era esperado e observou-se que as novas classes adicionadas à API de segurança Java são mais flexíveis e permitem um escopo maior de operações. Em particular, a geração de chaves mostrou-se bastante rápida e fácil. Esta operação também poderia ter sido executada por meio da ferramenta `keytool`, mas, a disponibilização de classes na API ajudaram que fossem obtidas implementações dos serviços de segurança.

Apesar das restrições relacionadas à compatibilização da MVJ dos *browsers* e das classes do kit JDK 1.2 serem um fato, obteve-se um mecanismo que pudesse efetivar a compatibilização entre ambas, através das ferramentas `policytool` e `HTMLConverter`.

VI.3 - TRABALHOS FUTUROS

A integração do serviço de Extrato de Conta Telefônica na Web a outros projeto em desenvolvimento no Laboratório de Redes e Gerência é o primeiro trabalho a ser executado.

Além disso, pretende-se que este trabalho seja continuado por outros alunos. Outros serviços de segurança podem ser implementados: auditoria,

confidencialidade, integridade e não repúdio. Os mecanismos principais para tal já foram implementados: criação e gerenciamento das chaves e assinatura digital, dessa forma, outras pessoas teriam condições de utilizá-lo como fonte.

A instalação da chave primária e de toda estrutura necessária para que o usuário possa cadastrar sua senha poderia ser feita por meio de um aplicativo que fosse entregue ao usuário por correio. O desenvolvimento deste aplicativo pode ser modificado em breve, haja visto que as versões previstas para os *browsers* já incluirão a MVJ para o JDK 1.2.

Um exemplo de implementação diferenciada das propostas deste trabalho seria a utilização de um Oracle Web Server [71]. A proposta de associar em um mesmo software a prestação de serviços Web habituais e os serviços de acesso a Banco de Dados torna o gerenciamento da segurança mais facilitado. Talvez, em um futuro próximo, o desenvolvimento de aplicações de Banco de Dados para a Web volte-se para esse paradigma, no entanto, ainda é cedo para afirmarmos tal fato, mas, é uma afirmação que pode servir para o início de uma nova pesquisa relacionada à temática dessa dissertação.

ANEXO A - CÓDIGO FONTE DA CLASSE CLIENT

```
import java.lang.*;
import java.awt.*;
import java.sql.*;
import java.io.*;
import java.security.*;
import java.applet.*;

public class Client extends Applet {

Label CPFL, nomeL, dataL, CEPL, enderecoL, cidadeL, estadoL;
TextField CPFTF, nomeTF, dataTF, CEPTF, enderecoTF, cidadeTF,
estadoTF;
Button exc, cad, mud;
Panel p1,p2;
PrivateKey priv;
PublicKey pub;

static {
            try {
            Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
            } catch (ClassNotFoundException e) {
             e.printStackTrace();
            }
           }

public static void main (String args[]) {
      Client teste = new Client();
      teste.init();
}

public void init () {
      setBackground (Color.black);
      setForeground (Color.white);

      p1 = new Panel();
      p2 = new Panel();

      cad = new Button ("Cadastre-me !");
      exc = new Button ("Exclua-me !");
      mud = new Button ("Quero mudar minha senha");

      CPFL = new Label ("CPF:");
      nomeL = new Label ("Nome:");
      enderecoL = new Label ("Endereco: ");
      cidadeL = new Label ("Cidade: ");
      estadoL = new Label ("Estado: ");
      CEPL = new Label("CEP: ");
      dataL = new Label ("Data:");
```

```
        CPFTF = new TextField (20);
        nomeTF = new TextField (20);
        enderecoTF = new TextField (20);
        cidadeTF = new TextField (10);
        estadoTF = new TextField (2);
        CEPTF = new TextField (10);
        dataTF = new TextField (10);

        p1.add (cad);
        p1.add (exc);
        p1.add (mud);


        p2.setLayout (new GridLayout (8,7));

        p2.add(CPFL);
        p2.add(CPFTF);
        p2.add(nomeL);
        p2.add(nomeTF);
        p2.add(enderecoL);
        p2.add(enderecoTF);
        p2.add(cidadeL);
        p2.add(cidadeTF);
        p2.add(estadoL);
        p2.add(estadoTF);
        p2.add(CEPL);
        p2.add(CEPTF);
        p2.add(dataL);
        p2.add(dataTF);

        add(p2,"North");
        add(p1,"South");

        setSize (500,300);
        show();
}

public boolean handleEvent (Event evt) {
        Object pEvtSource = evt.target;
        if (pEvtSource == exc && evt.id == Event.ACTION_EVENT) {
              return this.excDados ();
        }
               try {
                 this.url = new URL
("file:///d|/alex/extrato/apagou.html");
                  getAppletContext().showDocument(this.url);
                  }
               catch (MalformedURLException e)
              {
                System.out.println("Bad URL: " + this.url);
               }
```

```
      else if (pEvtSource == cad && evt.id == Event.ACTION_EVENT)
{
            return this.cadDados();
             try {
               this.url = new URL
("file:///d|/alex/extrato/ok.html");
               getAppletContext().showDocument(this.url);
               }
             catch (MalformedURLException e)
            {
              System.out.println("Bad URL: " + this.url);
             }
      }
      else if (pEvtSource == mud && evt.id ==
Event.WINDOW_DESTROY) {
            return this.mudDados();
             try {
               this.url = new URL
("file:///d|/alex/extrato/mudou.html");
               getAppletContext().showDocument(this.url);
               }
             catch (MalformedURLException e)
            {
              System.out.println("Bad URL: " + this.url);
             }

      }
      else if (pEvtSource == this && evt.id ==
Event.WINDOW_DESTROY) {
            return this.sair();
      }
      return false;
}

public boolean sair() {
      setVisible (false);
      System.exit (0);
      return true;
}
public boolean excDados() {

String CPF = CPFTF.getText().trim();

try {

            Connection con =
DriverManager.getConnection("jdbc:odbc:conta", "conta",
"conta");
            Statement stmt = con.createStatement();
            ResultSet rs = stmt.executeQuery("DELETE * FROM conta
Where (CPF ="'"+CPF+"')");

            rs.close();
```

```
            stmt.close();
            con.close();

            CPFTF.setText (CPF);
            nomeTF.setText (Name);
            enderecoTF.setText (Ender);
            cidadeTF.setText (City);
            estadoTF.setText (State);
            CEPTF.setText (CEP);
            dataTF.setText (Date);

      } catch (Exception e) {
        System.out.println ("Erro fatal na exclusão");
      }
      return true;
}

public boolean mudDados() {

String CPF = CPFTF.getText().trim();
String Name = nomeTF.getText().trim();
String Ender = enderecoTF.getText().trim();
String City = cidadeTF.getText().trim();
String State = estadoTF.getText().trim();
String CEP = CEPTF.getText().trim();
String Date = dataTF.getText().trim();

try {

            Connection con =
DriverManager.getConnection("jdbc:odbc:conta", "conta",
"conta");
            Statement stmt = con.createStatement();
            ResultSet rs = stmt.executeQuery("SELECT * FROM conta
Where (CPF ="'"+CPF+"')");

            while (rs.next())
            {
                  CPF = rs.getString(1);
                  Name = rs.getString(2);
                  Ender = rs.getString(3);
                  City = rs.getString(4);
                  State = rs.getString(5);
                  CEP = rs.getString(6);
                  Date = rs.getString(7);
            }
            rs.close();
            stmt.close();
            con.close();

            CPFTF.setText (CPF);
            nomeTF.setText (Name);
            enderecoTF.setText (Ender);
```

```
            cidadeTF.setText (City);
            estadoTF.setText (State);
            CEPTF.setText (CEP);
            dataTF.setText (Date);

      } catch (Exception e) {
        System.out.println ("Erro fatal na consulta");
      }
      return true;
}


public boolean cadDados() {

 try {
  KeyPairGenerator keyGen = KeyPairGenerator.getInstance ("DSA",
"SUN");
  SecureRandom random = SecureRandom.getInstance ("SHA1PRNG",
"SUN");

  KeyPair pair = keyGen.generateKeyPair();
  priv = pair.getPrivate();
  pub = pair.getPublic();

} catch (Exception e) {
  System.err.println ("Exceção ativada: " + e.toString());
}

        try {

           String CPF = CPFTF.getText().trim();
           String Name = nomeTF.getText().trim();
           String Ender = enderecoTF.getText().trim();
           String City = cidadeTF.getText().trim();
           String State = estadoTF.getText().trim();
           String CEP = CEPTF.getText().trim();
           String Date = dataTF.getText().trim();

           Connection con =
DriverManager.getConnection("jdbc:odbc:conta", "conta",
"conta");
           Statement stmt = con.createStatement();

           ResultSet rs = stmt.executeQuery("INSERT INTO conta
(CPF,nome,endereco,cidade,estado,CEP,data,chvp,chvpb) values
('"+CPF+"','"+Name+"','"+Ender+"','"+City+"','"+State+"','"+CEP+
"','"+Date+"','"+priv+"','"+pub+"')");
           stmt.close();
            con.close();
           System.out.println ("Operacao bem sucedida ! ");
      }
      catch (Exception e) {
      System.out.println ("Erro!");
```

```
        }

        CPFTF.setText ("");
         nomeTF.setText ("");
        enderecoTF.setText ("");
        cidadeTF.setText ("");
        estadoTF.setText ("");
        CEPTF.setText ("");
        dataTF.setText ("");


        return true;
}

}
```

ANEXO B - CÓDIGO FONTE DA CLASSE ASSINA

```
/* essa é a aplicação que o usuário vai utilizar em casa. A
política será a seguinte: em casa, o usuário recebe sua chave
privada e sua assinatura para uma senha sugerida. De posse dessa
assinatura, o usuário fornece a senha que é assinada e
verificada localmente pelo servidor que já possui todos os dados
*/

import java.lang.*;
import java.awt.*;
import java.sql.*;
import java.io.*;
import java.net.*;
import java.security.*;
import java.security.spec.*;
import java.applet.*;

// o usuário pode cadastrar sua senha e assiná-la, usando a
chave privada

public class acesso extends Applet {

protected Label CPFL, senhaL;
protected TextField CPFTF, senhaTF;
Button Ok, Cancel;
Panel p1,p2;
URL url;
String realsign,sig;
PrivateKey priv;

static {
            try {
            Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
            } catch (ClassNotFoundException e) {
             e.printStackTrace();
            }
           }

public static void main (String args[]) {
     Decasa teste = new Decasa();
     teste.init();
}

public void init () {

     // setLayout (new GridLayout(2,2));
     setBackground (Color.black);
     setForeground (Color.white);

     p1 = new Panel();
```

```
      p2 = new Panel();

      Ok = new Button ("OK ");
      Cancel = new Button ("Cancel ");

      CPFL = new Label ("CPF:");
      senhaL = new Label ("Senha:");


      CPFTF = new TextField (20);
      senhaTF = new TextField (20);
      senhaTF.setEchoCharacter ('*');

      p1.add (Ok);
      p1.add (Cancel);


      p2.setLayout (new GridLayout (3,2));

      p2.add(CPFL);
      p2.add(CPFTF);
      p2.add(senhaL);
      p2.add(senhaTF);

      add(p2,"North");
      add(p1,"South");

      senhaTF.requestFocus();
      setSize (500,300);
      show();
}

public boolean handleEvent (Event evt) {
      Object pEvtSource = evt.target;
      if (pEvtSource == Ok && evt.id == Event.ACTION_EVENT) {
            return this.assina ();
      }
      else if (pEvtSource == Cancel && evt.id ==
Event.ACTION_EVENT) {
             CPFTF.setText("");
            senhaTF.setText("");
            try {
               this.url = new URL
("file:///d|/alex/extrato/pitti.html");
               getAppletContext().showDocument(this.url);
               }
             catch (MalformedURLException e)
            {
              System.out.println("Bad URL: " + this.url);
             }
      }
      else if (pEvtSource == this && evt.id ==
Event.WINDOW_DESTROY) {
```

```
            return this.sair();
      }
      return false;
}

public boolean sair() {
      setVisible (false);
      System.exit (0);
      return true;
}

public boolean assina() {

/* Para que a assinatura seja enviada, será necessária a chave
privada que gerará a assinatura. Essa informação está disponível
no BD signus.
Esse BD, é automaticamente gravado em c:\Wtelec, quando da
instalação do aplicativo pelo cliente.

Por outro lado, a assinatura gerada fica guardada na variável
realSig. Esse procedimento permite que o host servidor da World
Telecom confirme a autenticidade do usuário. */

String gosign = senhaTF.getText().trim();
/* gosign é a senha, ou seja, o item que será assinado para
posterior verificação*/

try {
      Connection con =
DriverManager.getConnection("jdbc:odbc:signus", "signus",
"signus");
      Statement stmt = con.createStatement();

            ResultSet rs = stmt.executeQuery("Select * From
signus ");
            while (rs.next())
            {
                  priv = rs.getString(1);
            }

            rs.close();
            stmt.close();
             con.close();
      }
      catch (Exception e) {
      System.out.println ("Erro!");
      }

try {
      Signature dsa = Signature.getInstance ("SHAwithDSA",
"SUN");
      dsa.initSign (priv);
```

```
      byte[] b1;
      gosign.getBytes(0, gosign.length(),b1, 0 );

      /* como o método update apenas recebe objetos byte[] como
entrada, é feita conversão antes */

      dsa.update (b1);

      byte[] assinatura = dsa.sign();
      // a assinatura enfim, é realizada

      sig = new String(assinatura,0);
      /* sig é transformada em string para ser comparada a
assinatura é comparada com o a assinatura já armazenada no
servidor da WTelecom */
      }
      catch (Exception e) {
        System.err.println ("Caught exeception "+ e.toString());
      }

try {
      String procura = CPFTF.getText().trim();

      Connection con =
DriverManager.getConnection("jdbc:odbc:conta", "conta",
"conta");
      Statement stmt = con.createStatement();

            ResultSet rs = stmt.executeQuery("Select * From conta
Where (CPF = '"+procura+"')");
            while (rs.next())
            {
                  realsign = rs.getString(10);
            }

            rs.close();
            stmt.close();
             con.close();
      }
      catch (Exception e) {
      System.out.println ("Erro!");
      }

      if (sig != realsign)
      {
      try {
                  this.url = new URL
("file:///d|/alex/extrato/pitti.html");
                  getAppletContext().showDocument(this.url);
                 senhaTF.setText ("");
               }
       catch (MalformedURLException e)
      {
```

```
                System.out.println("Bad URL: " + this.url);
        }
       }
       else
            {
       try {
                this.url = new URL
("file:///d|/alex/extrato/ok.html");
                getAppletContext().showDocument(this.url);
                senhaTF.setText ("");
           }
        catch (MalformedURLException e)
       {
                System.out.println("Bad URL: " + this.url);
        }
       }


       CPFTF.setText ("");
       senhaTF.setText ("");


       return true;
}

}
```

VII - BIBLIOGRAFIA

[01] G. McGraw and E. Felten. ***Java™ Security. Hostile Applets, Holes, and Antidotes.*** Wiley Computer Publishing. New York. 1997.

[02] D. Wallach, D. Balfanz, D. Dean and E. Felten. ***Extensible Security Architectures for Java.***. Proceedings of the 16th Symposium on Operating Systems Principles. Saint-Malo. 1997. pp 116-128. (http://www.cs.princeton.edu/sip/sosp97.html).

[03] C. Matayoshi e W. Ruggiero. ***Modelo de Segurança da Linguagem Java***. Anais do XVI Simpósio Brasileiro de Redes de Computadores. Niterói-RJ. 1998. pp. 577-595.

[04] Sun Microsystems. ***JDK 1.2 Security Architecture***. (http://java.sun.com/products/jdk/1.2/docs/guide/security/index.html).

[05] D. Dean, E. Felten and D. Wallach. ***Java Security: From HotJava to Netscape Beyond.*** (http://www.cs.princeton.edu/sip).

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

[07] G. Coulouris et allii. ***Distributed Systems: Concepts and Design.*** Addison-Wesley. USA. 1994.

[08] R. Orfali, D. Harkey and J. Edwards. ***The Essencial Distributed Objects Survival Guide.*** Ed. John & Wiley. 1996.

[09] A. Dogac, C. Dengi and T. Özsu. ***Distributed Object Computing Platforms.*** in *Communications of the ACM*. Vol. 41. USA. 1998.

[10] T. Egamal and S. Cotter. ***Netscape Security - open-standard solutions for the enterprise.*** 1998. (http://developer.netscape.com/docs/manuals/security/scwp/index.htm).

[11] Microsoft Corporation. ***Microsoft.com Statement of Privacy.*** (http://www.microsoft.com/misc/privacy.htm).

[12] Visigenic. ***Visibroker Gatekeeper Guide Reference.***

[13] A. Dogac et allii.. ***METU Interoperable Database System***.

(http://www.srdc.metu.edu.tr).

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

[15] Z. Tari. ***Using agents for secure access to data in the Internet.*** IEEE Comunications. 1997. pp.136-140.

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

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

[18] P. Dowd and J. McHenry. ***Network Security: It's time to take it seriously.*** in IEEE Network Computer Special Edition in Network Security. 1998. pp. 24-28.

[19] R. Oppliger. ***Security at the Internet Layer.*** in IEEE Network Computer Special Edition in Network Security. 1998. pp. 43-47.

[20] OMG Document number 98-12-09. ***Security Service Specification in CORBAservices: Common Object Services Specification.*** 1998. (http://www.omg.org/techprocess/meetings/schedule/Technology_Adoptions.html).

[21] D. Chizmadia. ***A Quick Tour of the CORBA Security Service***. (http://www.omg.org/news/corbasec.htm#[2]).

[22] E. Evans and D. Rogers. ***Using Java Applets and CORBA for Multi-User Distributed Applications***. in IEE Internet Computing. USA. 1997. pp. 43-55.

[23] M. Rees. ***Converting Web Applications to Java***. Proceedings of the Third Australian World Wide Web Conference. Lismore. Austrália. 1997.

[24] M. Biesbrouck. ***CGI Security Tutorial***. (http://www.csclub.uwaterloo.ca/u/mlvanbie/cgisec/).

[25] F. Oliveira. ***Segurança em Sistemas: seu CGI é seguro ?*** Revista da Informação e Tecnologia do Centro de Computação da Unicamp.

(http://www.revista.unicamp.br/revista/navegacao/index9.html).

[26] A. Rubin and D. Geer Jr.. ***A Survey of Web Security.*** in IEE Internet Computing. USA. 1997. pp. 34-41.

[28] Microsoft Corporation. ***Internet Server API Overview.*** (http://www.microsoft.com/WIN32DEV/APIEXT/ISAPIMRG.HTM).

[29] E. Ly. ***Distributed Java Applets for Project Management on the Web.*** in IEEE Computer. 1997. pp. 21-26.

[30] L. Korba. ***Towards Securing Network Management Agent Distribution and Communication.*** to be published in the Proceedings of IM'99. Boston - MA. USA. 1999.

[31] R. Orfali and D. Harkey. ***Client/Server programming with Java and CORBA.*** New York. Ed. John & Wiley. 1997.

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

[33] S. Zeiger. ***Servlet Essentials.*** (http://www.novocode.com/doc/servlet-essentials/).

[34] Sun Microsystems. ***Servlet API Specification.*** (http://java.sun.com/products/servlet/2.1/html/servletapiTOC.fm.html)

[35] V. Yang. ***From Servlets to Enterprise Server.*** (http://www.servletcentral.com/1999-03/enterprise.dchtml)

[36] C. Dalton. ***Writing Good Servlets: an Overview.*** (http:// www.servletcentral.com/1999-01/design.dchtml)

[37] T. Stockwell. ***RMI on the Web.*** (http:// www.servletcentral.com/1999-03/rmi.dchtml)

[38] C. Hawblitzel, C. Chang, G. Czajkowski, D. Hu, and T. von Eicken. ***Implementing Multiple Protection Domains in Java.*** Proceedings of the 1998 USENIX Annual Technical Conference New Orleans, LA, June 1998. (http://www.cs.cornell.edu/SLK/jk-0.91/doc/Usenix98/usenix98-final.html).

[39] S. Coy. ***Security Implications of the choice of Distributed Management System Model: Relational vs Object-Oriented.***

Proceedings of 19th National Information Security Conference in http://jya.com/nissc96.htm.

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

[41] R. Elmasri and S. B. Navathe. ***Fundamentals of Database Systems.*** The Benjamin Cummings Publishing Company. Vancouver-Canadá. 1994.

[42] Microsoft Corporation. ***Cookies: What They Are, Why You Are In Charge.*** (http://register.microsoft.com/regwiz/include/cookies.htm).

[43] Netscape. ***Persistent Client State Http Cookies.*** (http://home.netscape.com/newsref/std/cookie_spec.html).

[44] Banco do Brasil. ***Personal Home Banking.*** (http://www.bancobrasil.com.br/personal/).

[45] Visa. ***Privacy Matters.*** (http://www.visa.com/cgi-bin/vee/ut/faq/privacy.html?2+0).

[46] Amazon Books. ***Why is Amzon.com so safe ?*** (http://www.amazon.com/exec/obidos/subst/help/security-guarantee.html/002-4580408-3174238).

[47] I. Nunes. ***O Ambiente Web Banco de Dados: Funcionalidades e Arquiteturas de Integração***. Dissertação de Mestrado. Departamento de Informática. Puc-Rio. 1997.

[48] P. F. Pires. ***Himpar, uma arquitetura para inteorperabilidade de Objetos Distribuídos***. Dissertação de Mestrado. Pós-Graduação em Engenharia de Sistemas e Computação. Universidade Federal do Rio de Janeiro. Rio de Janeiro - R.J. 1997.

[49] E. Bina, V. Jones, R. McCool and M. Winslett. ***Secure Access to Data Over the Internet.*** Proceedings of the Third ACM/IEEE International Conference on Parallel and Distributed Information Systems. Austin. Texas. 1994. (http://bunny.cs.uiuc.edu/CADR/WinslettGroup/pubs/SecureDBAccess.ps)

[50] J. Stabile, R. Pang and M. Anand. ***An Authentication Model for a Web Application Server***. Oracle Corporation. 1996.

(http://www.olab.com/www6_2)

[51] D. Rhamel. ***Systems and Methologies for Identifying and Protecting Weak Spots in Your Web-Enabled Database.*** Internet Systems. 1997. (http://www.dbmsmag.com/9704i03.html)

[52] C. Varela and C. Hayes. ***Zelig: Schema-based Generation of Soft WWW Database Applications.*** Proceedings of the Third International Conference on the World Wide Web. CERN. Geneva. Switzerland. 1994. (http://fiaker.ncsa.uiuc.edu:8080/WWW94.html.)

[53] C. Varela and C. Hayes. ***Providing Data on the Web: From Examples to Programs.*** Proceedings of the Second International Conference on the World Wide Web. Chicago. USA. 1994. (http://www.ncsa.uiuc.edu/SDG/IT94/Proceedings/DDay/varela/paper.html)

[54] M. Frank. ***Database and the Internet.*** DBMS On-Line Magazine. 1995. (http://www.dbmsmag.com/f19512.html)

[55] K. North. ***Building Web Databases.*** Web Techiniques Magazine. Vol 1(6) 1996. pp. 34-41.

[56] K. Reichard. ***Web servers for Database Applications.*** Internet Systems. 1996. (http://www.dbmsmag.com/9610i08.html)

[57] National Center for Supercomputing Applications. ***Common Gateway Interface Overview.*** University of Illionois. USA. 1997. (http://hoohoo.ncsa.uiuc.edu/cgi/overview.html).

[58] J. Levitt. ***Internetview: An Application Infrastructure.*** Information Week. 1996. (http://techwev.cmp.com/iw/582/82iojl.html).

[59] F. De Paoli, A. Santos and R. Kemmerer. ***Web Browsers and Security.*** in Mobile Agents and Security. Springer-Verlag. Editor G. Vigna. LNCS 1419. pp 235-256. Berlim. 1998.

[60] L. Gong et allii. ***Implementing Protection Domains in the Java Development Kit 1.2.*** Proceedings of the Internet Society Symposium on Network and Distributed System Security . Monterrey. California. 1997.

[61] CertiSign. ***Certificação Digital.*** (http://www.certisign.com.br)

[62] R. Stevens. ***TCP/IP Ilustrated, Volume 3: TCP for Transactions, HTTP, NNTP and the Unix Domain Protocols***. Addison-Wesley. 1996.

[63] Apache Digital Corporation. ***Apache and Secure Transactions***. 1998. (http://www.apacheweek.com/features/ssl).

[64] R. Smith. ***Internet Criptography***. Addison-Wesley. 1997.

[65] C. Semeria. ***Internet Firewalls and Security***. 3COM Corporation. 1996. (http://www.3com.com).

[66] D. Martin, S. Rajagopalan and A. Rubin. ***Blocking Java Applets at the Firewall***. Proceedings of the IntenetSociety Symposium on Network and Distributed System Security. 1997.

[67] J. Udell. ***Java Servlets***. Byte Magazine. June 1997. pp. 115-118.

[68] P. Wayner. ***Who Goes There ?*** Byte Magazine. June 1997. pp. 70-80.

[69] P. Diwanji, D. Conelly and P. Wagle. ***Java Server and Servlets.*** Proceedings of the JavaOne Conference'96. (http://java.sun.com/javaone/javaone96/pres/ServExt.pdf)

[70] Sun MicroSystems. ***Java Web Server***. 1995. (http://jeeves.javasoft.com).

[71] R. Muller. ***Oracle's Web-Footed Friend***. Byte Magazine. June 1997. pp. 141-142.