UNIVERSIDADE FEDERAL DE SANTA CATARINA

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Ubirajara Maia de Oliveira

INFRA-ESTRUTURA DE UM SERVIDOR DE APLICAÇÃO WEB SEGURO

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

Prof. Dr. Vitório Bruno Mazzola

Florianópolis, Maio de 2002.

INFRA-ESTRUTURA DE UM SERVIDOR DE APLICAÇÃO WEB SEGURO

Ubirajara Maia de Oliveira

Esta Dissertação foi julgada adequada para a obtenção do título de Mestre em Ciência da Computação Sistemas de Computação e aprovada em sua forma final pelo Programa de Pós-Graduação em Ciência da Computação.

Prof. Dr. Vitório Bruno Mazzola

Prof. Dr. Fernando Ostuni Gauthier

Banca Examinadora

Prof. Dr. João Bosco Mangueira Sobral

Prof. Dr. João Bosco da Mota Alves

Prof. Dr. Mauro Roisenberg

"A luta contra o erro
tipográfico tem algo de
homérico. Durante a revisão
os erros se escondem, fazem-
-se positivamente invisíveis.
Mas assim que o livro sai,
tornam-se visibilíssimos..."
(Monteiro Lobato)

Ofereço esta dissertação aos meus pais.

Agradecimentos

A Deus, sempre presente em minha vida.

À Carla Maria pela dedicação e companheirismo.

Aos amigos Glauco Vinicius Schaefel e Joaquim Cabral da Maia Neto pelo apoio e consultoria.

Aos colegas e familiares que sempre incentivaram a conclusão da dissertação.

À Universidade Federal de Santa Catarina pela infra-estrutura oferecida.

Aos membros da banca examinadora pela paciência e dedicação na leitura da dissertação de mestrado.

Em especial ao meu orientador Mazzola pela oportunidade, foco e correção de rumos, sem os quais a realização deste trabalho não seria possível.

Sumário

Lista de Tabelas

Lista de Figuras

Resumo

Este trabalho constitui-se de estudo sobre o uso de servidores de aplicação Web para suportar negócios na Internet, uma vez que se trata de forte tendência de mercado e devem rapidamente tornar-se padrão para todas as empresas. O estudo abrange a arquitetura e os mecanismos de segurança envolvidos, de maneira que seja possível garantir a integridade das transações financeiras realizadas entre as organizações, seus fornecedores e seus clientes. Resumem-se conceitos de criptografia, tecnologia Java J2EE, infra-estrutura do servidor de aplicação Web, e por último, propõe-se à análise mais detalhada de um destes componentes.

Palavras-chave: Servidor de aplicação Java, criptografia, segurança na internet.

Abstract

This dissertation consists in the Web application server use to support businesses through the Internet, that in turn constitute a strong market tendency that must quickly become standard for all the companies. The text covers architecture and security mechanisms involved to guarantee financial transactions integrity between organizations , its suppliers and its customers. It also presents concepts of cryptography, Java J2EE technology, application server infrastructure, and a detailed analysis of one of these components.

Keywords: Java application server, cryptography, internet security.

1 Capítulo - Introdução

1.1 Considerações Iniciais

Poucas tecnologias revolucionaram mais os negócios que o advento da Internet. Desde os meados da década de 90 empresas em todo o mundo perceberam rapidamente que o valor real da Internet não se resume na habilidade das pessoas em navegar pela rede ou em enviar e-mails, mas nas novas oportunidades que proporcionou para melhorar os processos dos negócios, reduzindo custos e aumentando lucros.

Os servidores de aplicação Web estão em via de se tornar padrão para qualquer pessoa ou empresa que queira fazer negócios pela Internet. Conforme apresentado na fig. 1, esse produto movimenta em torno de 3 bilhões de dólares anuais; em 2004, a cifra deve passar dos 11 bilhões, estimam os institutos de pesquisa Giga Information Group e Gartner Group.

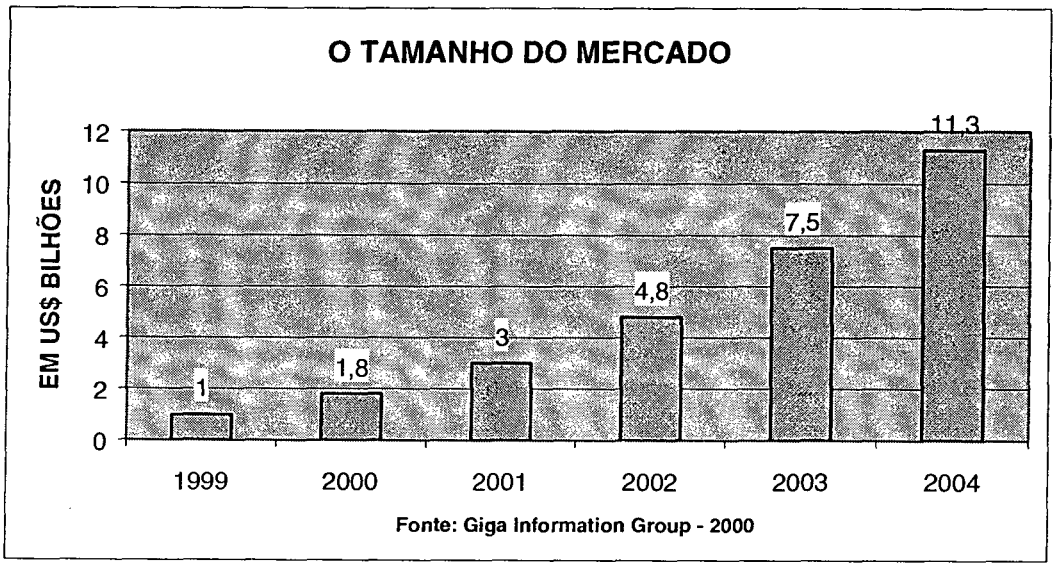

FIGURA 1 - O TAMANHO DO MERCADO

Em linhas gerais, o servidor de aplicação facilita a construção, manutenção e funcionamento de um *site* de negócios na Internet. O maior propósito de um servidor de aplicação é reduzir a carga de trabalho das aplicações tomando a responsabilidade por funções secundárias envolvidas na execução da aplicação e disponibilizando serviços para componentes externos de uma maneira confiável.

O mercado é dominado por algumas poucas empresas, conforme apresentado na figura 2, e tem a tendência de ser composto, em um breve futuro, por pequeno número de fornecedores de *software*.

Ainda é reduzido o número de empresas que se beneficiam desta tecnologia: menos de 10% das empresas utilizam esse tipo de programa, diz a empresa de pesquisas Giga Group. Os índices de crescimento seguem a velocidade da Internet.

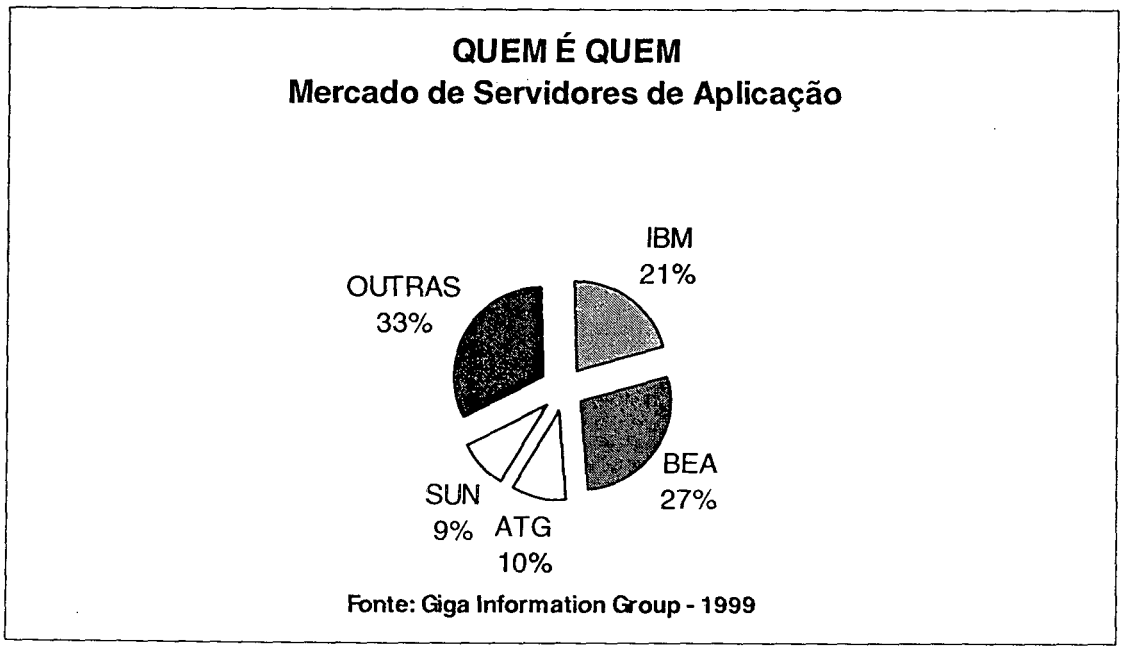

FIGURA 2 - QUEM É QUEM NO MERCADO DE SERVIDORES DE APLICAÇÃO.

Para obter sucesso neste novo cenário, as empresas precisam criar uma infra-estrutura de tecnologia da informação que seja otimizada para suportar as necessidades do mercado que agora emerge. Uma infra-estrutura de TI é o conjunto de ferramentas que coloca a empresa na Internet. Enquanto as ferramentas necessárias para apoiar o processo de um negócio podem variar entre as empresas, uma infra-estrutura geralmente é consistente e comporta os seguintes componentes:

- Infra-estrutura de rede;
- Infra-estrutura de segurança;
- Ambiente do servidor da aplicação;
- Ferramentas de controle dos dados e do conteúdo;
- Ferramentas de desenvolvimento das aplicações;
- Máquinas e sistemas operacionais;
- Plataforma de gerenciamento dos sistemas.

As exigências de segurança também estão mudando. Ainda que muitas empresas tenham implementado sistemas básicos de autorizações por meio do uso de senhas, os mais complexos sistemas na Internet requerem um ambiente mais sofisticado além do tradicional *firewall*. Este fato faz com que as funções de segurança passem de simples senhas para certificados digitais de segurança, os quais não apenas proporcionam a validação individual do usuário, como também fornecem diferentes níveis de validação, dependendo da aplicação ou dos dados a serem acessados.

A próxima etapa na segurança é fornecer a segurança com base em normas de controle. Um servidor de controle das normas fornece autorizações de acesso único e global para sistemas múltiplos que podem eliminar a necessidade de várias senhas.

Pode também controlar a segurança de forma independente em cada uma das aplicações individuais, com a autorização administrada e controlada em um único local.

Ao utilizar sistema de segurança baseado em normas, as empresas podem criar uma técnica em camadas para obter segurança, oferecendo maiores graus de proteção contra acessos não autorizados sem que o sistema se torne um problema desnecessário para os usuários credenciados.

1.2 Estrutura do Trabalho

O estudo encontra-se assim organizado: este capítulo analisa o panorama do mercado de servidores de aplicação Web e a importância da segurança nas aplicações de negócio.

O capitulo 2 apresenta a fundamentação teórica sobre Criptografia e a tecnologia Java.

No capítulo 3 apresenta-se a infra-estrutura e os componentes de um servidor de aplicação Web.

O capítulo 4 explica detalhadamente os aspectos de segurança de um dos componentes selecionados.

O capítulo 5 apresenta a conclusão dos estudos e, conseqüentemente, as propostas de melhorias, objetivo desta dissertação de mestrado.

2 Capítulo - A Tecnologia dos Servidores de Aplicação

2.1 Criptografia

2.1.1 Introdução

Criptografia é a ciência de manter mensagens seguras utilizando algoritmos de criptografia. Criptografar é o processo de converter textos originais em criptogramas (textos criptografados), escondendo, desta forma, o conteúdo do texto original. Este texto original pode ser um fluxo de *bits*, um arquivo texto, uma imagem, um fluxo de voz – ou seja, qualquer informação digital ou digitalizada.

Descriptografar é o processo de voltar um texto criptografado (criptograma) ao texto original. Este processo geralmente requer um código ou chave. Uma chave representa uma seqüência de caracteres utilizados para codificar ou decodificar um arquivo.

A criptografia provê os seguintes serviços [SCHN96]:

Confidencialidade – constitui, sem dúvida, o serviço mais comum e a origem da criptografia. Permite que uma entidade A envie uma mensagem criptografada a uma entidade B e que nenhuma outra entidade consiga descriptar a mensagem.

Autenticação – permite que o transmissor tenha certeza de estar enviando a mensagem para o receptor correto, e vice-versa. Por exemplo, quando A envia a mensagem para B, B deve ter certeza de que a mensagem está realmente vindo de A.

Integridade – garante que o conteúdo da mensagem original seja preservado até o destino. Consequentemente, distorções nas mensagens são evitadas.

Não repudiação – garante que um transmissor não possa negar ter enviado uma certa mensagem. Ou seja, a partir do momento em que o receptor B recebeu uma mensagem do transmissor A, ele possui provas de que foi A quem realmente enviou a mensagem, e, por meio do serviço de integridade, possui provas de que a mensagem foi realmente aquela.

2.1.2 Algoritmos simétricos

Algoritmos simétricos são aqueles em que as chaves de criptografia e descriptografia são calculadas uma a partir da outra; em muitos algoritmos simétricos as chaves são as mesmas.

A utilização de algoritmos simétricos requer que o remetente e o destinatário concordem com uma chave antes de enviarem as mensagens, o que significa dizer que a segurança da comunicação depende do segredo desta chave. Se esta chave for conhecida, a mensagem pode ser decifrada.

2.1.2.1 DES

O DES (*Data Encryption Standard*), também conhecido como DEA (*Data Encryption Algorithm*) pelo ANSI e DEA-1 pelo ISO, é um algoritmo simétrico que utiliza a mesma chave para criptografia e descriptografia. Sendo um dos algoritmos mais usados para criptografia atualmente, foi desenvolvido em 1970 [GARF96] e patenteado em 15 de maio de 1973 pela IBM [NEWT97] sendo um aperfeiçoamento do

sistema *Lucifer*[1]. A IBM tornou o DES público em 1977 descrevendo o sistema, e, desde então, vem sofrendo revisões periódicas sendo a mais recente datada de dezembro de 1993 [GARF96]. A IBM originalmente usava uma chave de 128 *bits*, porém, após ser "convidada" pelo NSA (*National Security Agency*), a chave foi reduzida para 56 *bits* (e outros 8 bits foram usados para paridade). Especula-se que esta redução foi implementada para facilitar a quebra [TANE96,KAHN96]. As reuniões entre a IBM e NSA resultaram em uma melhoria da chave chamada *S-box* (onde o S indica substituição).

Recentemente o mundo presenciou turbulenta polêmica, na qual, de um lado encontravam-se a IBM (criadora do algoritmo) e a NSA, e, do outro lado, a comunidade acadêmica americana liderada por Martin Hellman e Whitfield Diffie, que acreditava que o NSA poderia ter acrescentado ao DES uma "trap door". No entanto, o DES foi adotado pelo NBS (*National Bureou of Standards*) em 1978, para uso pelos órgãos governamentais não militares nem diplomáticos, desde que sua implementação fosse em *hardware*, e sua utilização ficasse restrita à cifragem de dados não confidenciais. Um relatório da Lexar Corporation afirma que "Estruturas foram encontradas no DES que indubitavelmente foram inseridas para fortalecer o sistema contra determinados tipos de ataques. Também foram encontradas estruturas que aparentam enfraquecer o sistema" [SCHN96].

[1] Usado pelo Loyds Bank em 1973

2.1.2.2 RC4

Rivest Cipher 4, projetado por Ron Rivest na empresa RSA Data Security Inc., consiste em um algoritmo de criptografia para fluxo de dados com tamanho de chave variável. Assim como o RC2, existe uma possibilidade especial de exportação para este algoritmo[2], desde que as chaves não sejam maiores do que 40 *bits* [TANE96]. Este algoritmo foi mantido por sete anos como proprietário e os detalhes do algoritmo só eram disponibilizados após a assinatura de um contrato de *non-disclosure.* Em setembro de 1994 o código[3] foi postado anonimamente no *mailing-list Cypherpunks* e a partir daí ele se tornou altamente difundido ao redor do mundo [SCNH96].

2.1.3 Algoritmos assimétricos

2.1.3.1 Introdução

Algoritmos assimétricos ou algoritmos de chave pública, diferem dos algoritmos simétricos pela utilização de duas chaves: uma pública, disponível para todos que queiram enviar uma mensagem criptografada a um receptor B, e uma chave secreta (privada), diferente da chave pública, utilizada por B para descriptar a mensagem. Esta classe de algoritmos foi criada em 1976 pelos pesquisadores Whitfield Diffie e Martin Hellman na Universidade de Stanford [KAHN96,NEWT97], apesar de a NSA haver afirmado (sem apresentar provas) já conhecer este conceito [SCHN96].

---

[2] O governo americano jamais autorizaria a exportação se ele ao menos em teoria não puder quebrar. [SCHN96]

[3] Apesar disto, o código e o nome continuaram possuindo direitos autorais — no entanto muitas pessoas ao redor do número usam o código, com ou sem variações, sem citar o nome RC4.

As características deste algoritmo são [TANE96]:

- um texto criptografado com a chave pública pode ser descriptado com a chave secreta;

- é extremamente difícil deduzir a chave secreta a partir da chave pública ;

- não se pode quebrar a chave pública por um ataque utilizando textos originais.

2.1.3.2 RSA

O RSA é o algoritmo assimétrico mais popular utilizado para criptografia e assinaturas digitais. Criado em 1977 pelos professores do MIT, Ronald Rivest e Adi Shamir, e pelo professor da USC, Leonard Adleman, o algoritmo é baseado na dificuldade de achar dois fatores primos para um dado produto. É usado para criptografar informações e serve de base para sistemas de assinatura digital. O algoritmo usa duas chaves e o tamanho destas chaves neste sistema pode variar dependendo da implementação, sendo que chaves maiores são consideradas mais seguras.

O algoritmo é mais popular na Europa e no Japão do que nos EUA pois apesar de ser patenteado[4] neste país, o algoritmo pode ser usado sem pagamento de *royalties* em qualquer lugar do mundo exceto nos EUA uma vez que foi publicado antes que a patente tivesse sido registrada [GARF96,SCHN96,TANE96].

---

[4] Apesar de existirem controvérsias sobre o fato de se patentear séries de transformações matemáticas [GARF96]

2.1.3.3 Diffie-Hellmann

É um algoritmo assimétrico inventado em 1976 e utilizado principalmente para distribuição de chaves. Como a patente deste algoritmo expirou em 1997, ele está tornando-se bastante popular.

2.1.4 *Message digests*[5]

Técnica também conhecida como *checksum* ou *cryptographic hashcode* [GARF96] representa uma versão reduzida de uma mensagem criada pela aplicação de um algoritmo de *hash* a uma mensagem original de um remetente. O propósito de gerar um *message digest* é possibilitar ao remetente anexar sua chave privada para autenticar a mensagem, podendo então o destinatário separar eletronicamente o *message digest* da mensagem propriamente dita [NEWT97]. Um *hashcode* nada mais é do que um número produzido por uma função que é muito difícil de ser revertido [GARF96].

Uma função *hash/message digest* recebe uma entrada, normalmente de tamanho indefinido e produz um número que é significativamente menor que a mensagem. A função possui as seguintes características [GARF96]:

- relaciona-se de forma 1 para muitos com as mensagens;
- é determinística, ou seja, o mesmo valor de saída é produzido para entradas idênticas;
- o processo não pode ser revertido;

[5] A tradução em português é "compilações de mensagens". Todavia os autores preferem manter o original em inglês

- uma pequena alteração no texto de entrada gera uma grande diferença na saída, conseqüência da primeira propriedade.

Esse mecanismo pode ser usado como um sistema de autenticação para verificar o remetente de um dado documento. Pode-se usar este mecanismo por exemplo para saber se a cópia para a qual foi dado *download* da Internet é uma cópia inalterada do original; simplesmente é preciso recalcular o *message digest* do documento e comparar com o valor do original.

2.1.4.1 MD5

Trata-se de um dos algoritmos mais usados atualmente[6] para *message digest*, constituindo uma versão melhorada do MD4, criado por Ron Rivest. Apesar de mais complexo é similar em projeto e produz também um valor *hash* de 128 bits. Não é conhecida nenhuma técnica de ataque melhor do que a força bruta, sendo sua dificuldade baseada na dificuldade de realizar fatorações[7]. O algoritmo é caracterizado por [SCHN96]:

- Ser computacionalmente inviável encontrar duas mensagens que produzam o mesmo resultado *hash*;
- Possuir 4 iterações;
- Cada estágio possuir uma constante de soma;
- Cada iteração influenciar na iteração subseqüente;
- Ser menos simétrico que o MD4;

---

[6] Usado, por exemplo, por softwares de e-mail

[7] Para mais detalhes ver o funcionamento do RSA.

- Os passos 2 e 3 serem mais diferentes entre si do que eram no MD4;
- O *shift* realizado em cada iteração ser otimizado em relação ao MD4 para que ocorra mais rapidamente o efeito de propagação.

São detectadas algumas fraquezas na função de compressão, porém não possuem nenhum impacto prático sobre a segurança da função *hash* usada [SCHN96].

2.1.5 Assinatura digital

A autenticidade de muitos documentos legais e de outros documentos é determinada pela presença de uma assinatura autorizada. Para que os sistemas de mensagens computadorizadas possam substituir o transporte físico de documentos em papel e tinta, procurou-se encontrar uma solução para esse problema [TANE96].

Os requisitos para que se possa enviar uma mensagem são:

- O receptor possa verificar a identidade alegada pelo transmissor;
- Posteriormente o transmissor não possa repudiar o conteúdo da mensagem;
- Provavelmente o próprio receptor tenha inventado a mensagem.

2.2 A tecnologia Java

2.2.1 Introdução

Os maiores desafios e as oportunidades mais interessantes para os desenvolvedores de *software*, hoje em dia, giram em torno de aproveitar todo o poder

das redes. As aplicações criadas atualmente serão executadas em máquinas ligadas por uma rede global de recursos de computação. A importância cada vez maior das redes está impondo novas demandas sobre as ferramentas existentes e alimentando a demanda por uma lista cada vez maior de tipos de aplicações totalmente novos [KNUD00].

Java é uma linguagem de programação desenvolvida pela *Sun Microsystems* cujo objetivo foi criar uma linguagem dinâmica orientada a objetos para uso nos mesmos tipos de aplicações desenvolvidas em C e C++, mas sem as dificuldades e os erros mais comuns destas linguagens [KNUD00]. De acordo com a experiência relatada pelos projetistas da linguagem, foram realizadas escolhas que adicionaram novas características ou supriram outras do C++, visando criar uma linguagem que facilitasse a geração de código robusto, confiável e facilmente utilizável em plataformas diversas.

A linguagem Java fornece inúmeras bibliotecas de classes, tornando possível o desenvolvimento de aplicações completas apenas utilizando estas classes básicas (que formam o chamado *Java Core*). Como exemplo destas bibliotecas de classes têm-se: interface com redes; interface com segurança; interface com banco de dados (JDBC).

A tabela a seguir apresenta as principais APIs da arquitetura Java J2EE:

API	**Descrição**
EJB	*Enterprise JavaBeans* é um modelo de servidores de componentes que provêm portabilidade através de servidor de aplicações e implementa serviços automáticos em benefício dos componentes da aplicação.
JNDI	*Java Naming and Directory Interface* provê acesso a nomes e diretórios de serviços como DNS, NDS, LDAP, e nomes CORBA.
RMI/IIOP	*Remote Method Invocation* cria interfaces remotas para comunicação java-para-java. Esta extensão usa o protocolo de comunicação IIOP padrão CORBA.
Java IDL	*Java Interface Definition Language* cria interface remota para suportar comunicação Java-para-CORBA.

Servlets e JSP	*Java servlets* e *Java Server Pages* são servidores componentes que rodam em um Servidor Web que suporta a geração de HTML dinâmico e gerenciamento de sessão para navegadores clientes.
JMS	*Java Messaging Service* suporta comunicação assíncrona usando um mecanismo de enfileiramento confiável ou através do modelo publicar e subscrever.
JTA	*Java Transaction API* provê uma API de demarcação transações.
JTS	*Java Transaction Service* define um serviço gerenciamento de transações distribuídas baseados no Serviço de Objetos de transação CORBA.
JDBC	*JDBC Database Access API* provê acesso uniforme para banco de dados relacionais.
JavaMail	*JavaMail* provê um protocolo independente para construir mensagens e aplicações de correio eletrônico.
JAF	*JavaBeans Activation Framework* provê um padrão de serviços para determinar o tipo de pedaço arbitrário de dados e ativar componentes de *JavaBeans* apropriados para manipular dados.

TABELA 1 - ENTERPRISE JAVA APIS

A Máquina Virtual Java (JVM) é o principal meio pelo qual é atingida a independência de plataforma na linguagem Java [KNUD00]. O alvo do compilador não é o *assembler* de um processador específico, pois isto limitaria o código gerado a uma determinada arquitetura. O compilador gera o código aberto ou *bytecode* para um processador hipotético, para a Máquina Virtual Java. As implementações da JVM nas diversas combinações de plataformas de *hardware/software* possibilitam a execução de *bytecodes.* As primeiras JVMs implementam, na prática, um interpretador de *bytecodes*, acarretando, consequentemente, um pior desempenho na execução do código. Atualmente já estão disponíveis ambientes onde antes de serem executados, os *bytecodes* são compilados para o código objeto da plataforma de execução (são os chamados Compiladores de tempo real). Desta maneira, as perdas na velocidade de processamento são consideravelmente reduzidas.

Os pontos que tornam a linguagem mais robusta são destacados abaixo:

- Ausência de aritmética de ponteiros;
- Verificações em tempo de compilação;
- Verificações em tempo de execução;
- Algoritmo de *gabage collection.*

2.2.2 Criptografia aplicada em Java

2.2.2.1 Introdução

A segurança na linguagem Java é uma característica-chave da arquitetura da linguagem que a torna uma opção apropriada para ambientes de redes de computadores. Segurança é importante porque estes ambientes permitem um potencial ataque a partir de qualquer computador que tenha acesso à rede. Estas preocupações tornam-se especialmente fortes quando programas (*software*) são trazidos por meio da rede e executados localmente, como ocorre com *applets* Java. É provável que navegando pela rede, o usuário encontrará *applets* de origem não confiável. Para enfrentar este ambiente hostil, os mecanismos de segurança da linguagem Java estabelecem uma distinção de tratamento dependendo da origem do código a ser executado.

O modelo de segurança Java propõe-se a proteger o usuário de programas hostis trazidos pela rede de fontes não confiáveis. Para tanto, o ambiente Java utiliza-se de uma *sandbox* dentro do qual o programa Java é executado. O programa pode realizar qualquer procedimento dentro dos limites da *sandbox*, mas não consegue realizar nenhuma ação fora de seus limites. *Applets Java* trazidos a partir da rede não podem realizar inúmeras ações, entre as quais:

- Ler ou escrever no sistema de arquivos da máquina do usuário;
- Estabelecer conexões de rede para qualquer servidor, a não ser com o servidor do qual o *applet* foi carregado;
- Criar novos processos;
- Carregar dinamicamente novas bibliotecas e chamar diretamente métodos nativos.

2.2.2.2 Java Cryptography Architecture

Java Cryptography Architecture (JCA) (Arquitetura de Criptografia do Java) refere-se ao conjunto de classes e processos para que programas em Java possam utilizar-se de – e também desenvolver – funcionalidades de criptografia para a plataforma Java. É composta por parte da biblioteca de classes JDK *Java Security*, bem como uma série de convenções e especificações.

2.2.2.3 Diretivas de projeto

O projeto da JCA foi feito de acordo com um conjunto de diretivas:

- Independência de implementação: permite que para uma determinada funcionalidade (ex.: *message digest*) possa haver implementações de diferentes fornecedores sem que o desenvolvedor que utiliza esta funcionalidade tenha que se preocupar com estas diferentes implementações. Isto é feito por meio de interfaces padrões que todos os fornecedores devem seguir. Como parte do JDK

existe um fornecedor padrão de nome "SUN" com implementações do *Digital Signature Algorithm* (DSA), além de MD5 e SHA-1 para *message digests;*

- Interoperabilidade de implementação: permite que diferentes implementações possam trabalhar, utilizar chaves e verificar assinaturas entre si;
- Independência de algoritmos: os algoritmos de criptografia estão encapsulados em classes, as quais são denominadas *engine classes*, como por exemplo *MessageDigest* e *Signature*;
- Extensibilidade de algoritmos: novos algoritmos que possam ser adequados às *engine classes* podem ser facilmente incorporados.

2.3 Principais classes

2.3.1.1.1 Classe Provider

É uma classe abstrata para um pacote ou conjunto de pacotes que implementam algoritmos de criptografia específicos. Possui métodos para acessar o nome do provedor, número de versão e informações adicionais.

Pacotes fornecidos por provedores podem incluir implementações para os seguintes tipos de algoritmos:

- Algoritmos de assinatura digital: ex.: DSA ou MD5 com RSA;
- Algoritmos de message digest: ex.: MD5 ou SHA-1;
- Algoritmos de criptografia: ex.: DES ou RSA;
- Esquemas de preenchimento: ex.: PKCS#5.

2.3.1.1.2 Classe Security

Esta classe gerencia os provedores e centraliza as propriedades e métodos de segurança. É uma classe final, contendo apenas métodos estáticos e nunca é instanciada.

Os principais métodos desta classe são:

- public static Provider[] getProviders() – utilizado para obter um vetor com todos os provedores instalados;
- public static int addProvider(Provider provedor) – utilizado para instalar novos provedores em tempo de execução;
- public static void removeProvider(String nome) – utilizado para remover o provedor com o nome especificado.

2.3.1.1.3 Engine classes

As *engine classes* (*MessageDigest, Signature e KeyPairGenerator*) provêm a funcionalidade para calcular o *message digest* de uma mensagem, assinar e verificar assinaturas digitais e gerar pares de chaves públicas e privadas.

As *engine classes* são referenciadas por *Service Provider Interfaces* (SPIs). Há uma API correspondente a cada *engine class*, permitindo que desenvolvedores usem instâncias destas *engine classes* para criar aplicações que utilizam *message digests* ou assinaturas digitais.

Para cada *engine class* há um método fábrica disponível que retorna uma instância da classe. Por exemplo, o método *getInstance* (*String* algoritmo) de *Message Digest* (ou *Signature*) encontra uma subclasse de *Message Digest* (ou *Signature*) que satisfaça o algoritmo especificado (utilizando o provedor padrão).

2.3.1.1.4 Interface Key

Define a funcionalidade compartilhada por todos os objetos que são chave de criptografia, contendo como características: um algoritmo para a chave; um formato de codificação (que é o formato de codificação externa para chave, pois a chave é codificada utilizando um formato padrão como X.509 ou PKCS#8); e o nome do formato para a chave codificada.

2.3.1.1.5 Classe Signature

A classe *Signature* é utilizada para obter um objeto que pode ser usado para assinar dados ou verificar se uma certa assinatura é autêntica. É composta por:

- API de Assinatura Digital – interface utilizada pelos desenvolvedores que necessitam de serviços de assinatura digital;
- SPI de Assinatura Digital – interface implementada pelos provedores que oferecem algoritmos específicos. Todos os métodos desta interface possuem o nome prefixado por "*engine*".

```
// criação do objeto Signature
Signature dsa = Signature.getInstance("DSA");
// criação do objeto para gerar pares de chaves
KeyPairGenerator keyGen = KeyPairGenerator.getInstance("DSA");
// inicialização do gerador de pares de chaves
keyGen.initialize(1024, new SecureRandom(userSeed);
// geração do par de chaves
KeyPair pair = keyGen.generateKeyPair();
// obtenção da chave secreta (privada)
PrivateKey priv = pair.getPrivate();
```

```
// inicialização do objeto Assinatura para assinar
dsa.initSign(priv);
// atualização do objeto Assinatura com os dados a serem assinados
dsa.update(data);
// assinar os dados
byte[] sig = dsa.sign();
```

2.3.2 *Message digests*

O JDK possui a classe *message digest*, a qual provê uma interface para a funcionalidade de um algoritmo de *message digest*. Seguindo as diretivas do projeto da JCA, a implementação de algoritmos de *message digest* é de responsabilidade dos provedores. No JDK existe um provedor *default* denominado "SUN", que provê os algoritmos MD5 e SHA-1 para *message digests*.

Também seguindo as diretivas de projeto da JCA, há duas interfaces principais associadas a *message digests*: API de *Message Digest*, utilizada pelos desenvolvedores; e SPI de *Message Digest*, implementada pelos provedores.

Os principais métodos da classe *MessageDigest* são:

- public static **getInstance**(String algorithm) throws NoSuchAlgorithmException – gera um objeto *MessageDigest* que implementa o algoritmo especificado;
- public void update(byte input[]) – atualiza o objeto com os dados a ele associados;
- public byte[] digest() – calcula o *digest* para o objeto utilizando o algoritmo especificado na criação do objeto. O valor *hash* calculado para os dados é retornado como um vetor de *bytes*;

- public void reset() – volta o objeto ao seu estado inicial.

Exemplo de utilização da classe MessageDigest:

```
// criação de um objeto MessageDigest com o algoritmo MD5
MessageDigest md5 = MessageDigest.getInstance("MD5");

// leitura dos dados a serem "digested"
File arq = new File("carta0499.doc");
FileInputStream in = new FileInputStream(arq);
byte[] doc = new byte[(int) (arq.length())];
in.read(doc);
// cálculo do digest
md5.update(doc);
byte[] md5Digest = md5.digest();
```

2.3.3 Assinaturas digitais e certificados

Há duas ferramentas introduzidas no JDK 1.2 para gerenciar chaves e certificados e para assinatura de arquivos JAR:

- *keytool* – gerencia o banco de chaves secretas (*keystore*) e também a corrente de certificados[8] X.509 que autenticam suas respectivas chaves públicas. Também gerencia os certificados de entidades confiáveis;
- *jarsigner* – gerencia assinaturas para arquivos JAR e verifica as assinaturas de arquivos JAR assinados.

2.3.3.1 Keystore e keytool

O *keystore* é definido na classe abstrata *KeyStore*. Esta classe representa uma

[8] *certificate chains*, em inglês

coleção de chaves secretas e as correntes de certificados associadas para utilização em auto-autenticação. Estas chaves e correntes de certificados são utilizadas por uma certa entidade para autenticar a si mesma utilizando certificados de chave pública.

A implementação padrão (a que vem com o JDK) do *keystore* o implementa em um arquivo. A localização padrão é o arquivo .keystore no diretório indicado pela propriedade user.home. O *keystore* é protegido por uma senha. É possível que desenvolvedores escrevam sua própria implementação de *KeyStore*, por exemplo, para utilizar um formato diferente (ex.: banco de dados relacional).

O utilitário *keytool* é utilizado para gerenciamento de chaves e certificados com as seguintes funções: criação do *keystore*, inclusão de entradas no *keystore* (cada entrada é protegida por uma senha), alteração da senha de uma entrada, importação de certificados, exibição das entradas, exportação de um certificado, geração de um certificado auto-assinado.

O exemplo a seguir mostra como criar um *keystore*, associá-lo a uma senha e criar um certificado auto-assinado:

```
$ keytool -genkey
Enter keystore password: a25f18
What is your first and last name?
[Unknown]: Usuário Teste
What is the name of your organizational unit?
[Unknown]: PPG
What is the name of your organization?
[Unknown]: UFSC
What is the name of your City or Locality?
[Unknown]: Florianópolis
```

```
What is the name of your State or Province?
[Unknown]: SC
What is the two-letter country code for this unit?
[Unknown]: BR
Is <CN=Usuário Teste, OU=PPG, O=UFSC, L=Florianópolis, S=SC, C=BR> correct?
[no]: y
Enter key password for <mykey>
(RETURN if same as keystore password): a25f18key
```

Neste exemplo, foi criado um par de chaves pública/secreta e a chave pública é colocada em um certificado auto-assinado. Certificado auto-assinado é aquele em que a entidade que o assina é a mesma cuja chave pública está sendo autenticada pelo certificado.

2.3.4 Arquitetura Java J2EE

Um servidor de aplicação é um programa de computador que está instalado em um servidor em uma rede distribuída e sua principal função é prover a lógica de negócios para um programa, obtendo flexibilidade e customização no ambiente de execução para hospedar componentes de negócio, serviços distribuídos e integridade para execução da aplicação.

Os *Enterprise Java Beans* (EJB) são principalmente componentes no lado servidor para aplicações em rede reunindo APIs do Java orientadas para empresa (incluindo acesso a banco de dados, transações e serviços de nomes) em um único modelo de componente para aplicações de servidor. O EJB impõe muito mais estrutura sobre como escrever o código e permite que o *container* EJB no lado servidor assuma mais responsabilidade e otimize as atividades da aplicação sem que venha a escrever muito código. Algumas atividades que o EJB realiza:

- Ciclo de vida do Objeto e pesquisa em serviço de nomes;
- Persistência gerenciada pelo *container;*
- Gerenciador de Transação;
- *Pooling* e gerenciamento de recursos do servidor;
- Configuração da distribuição.

O EJB é dividido em dois campos: *beans* de entidade, que representam objetos e dados, e *beans* de sessão, que implementam serviços e operações sobre os *beans* de entidade. Estes correspondem à segunda e terceira camadas de uma aplicação comercial três camadas. A lógica de negócios é representada pelos serviços do *bean* de sessão e o acesso ao banco de dados se torna transparente por meio do mapeamento de objeto automatizado do *bean* de entidade.

Muitos aspectos do comportamento do EJB podem ser controlados através de "descritores de distribuição" que personalizam o comportamento do *bean* para o ambiente local. O resultado reflete um alto nível de abstração através do código comum, específico da empresa. Ele permite que poderosos componentes de aplicação comercial em rede sejam empacotados e realizados da mesma forma que os *beans* comuns são reutilizados para a criação de aplicações no lado do cliente.

O servidor de aplicação como o apresentado na fig. 3 compreende os servidores de *Enterprise Java Beans* e de *Java Server Pages*. Este é o responsável pela execução das classes Java e JSPs utilizados pelas aplicações.

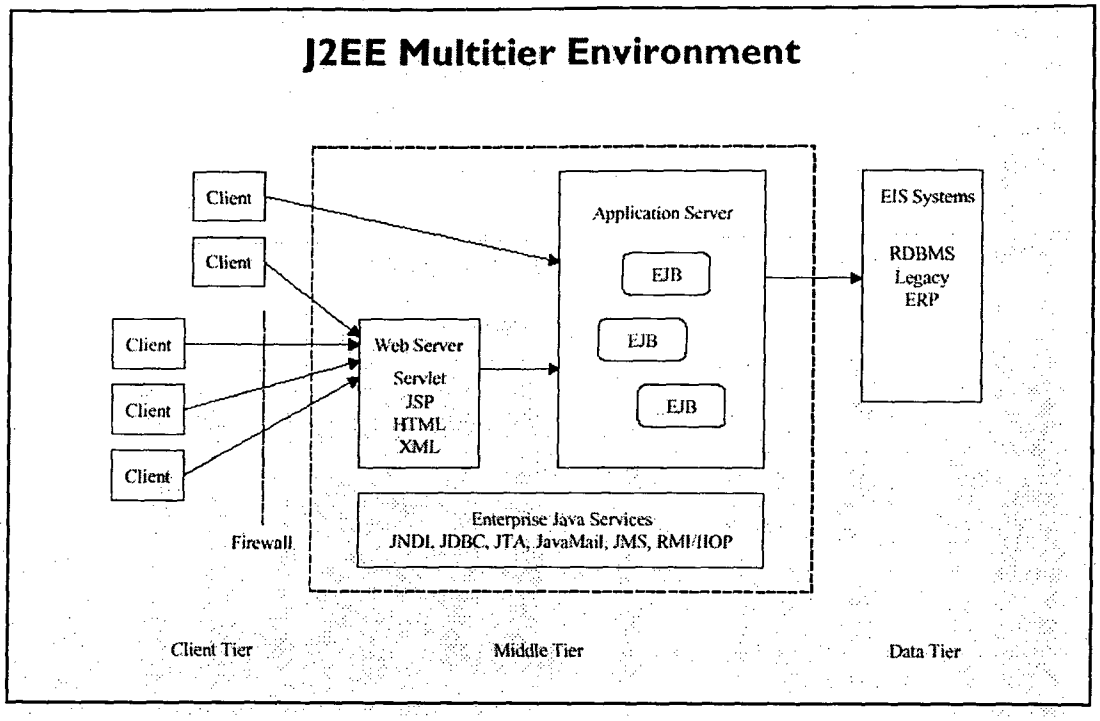

FIGURA 3 - ARQUITETURA J2EE MULTICAMADAS

2.3.4.1 Requisitos de segurança

Os requisitos de segurança previstos na arquitetura Java J2EE são descritos abaixo:

Portabilidade: A arquitetura de segurança deve suportar as propriedades de "escreva uma vez", rode em qualquer lugar;

Transparência: Os fabricantes de componentes de aplicação não deveriam ter que saber sobre segurança para escrever uma aplicação;

Isolamento: A plataforma J2EE deveria ser capaz de realizar autenticação e controle de acesso de acordo com instruções estabelecidas pelo desenvolvedor usando atributos de distribuição e gerenciado pelo administrador do sistema;

Extensibilidade: O uso de serviços da plataforma por aplicações seguras não deve comprometer a portabilidade da aplicação;

Flexibilidade: Os mecanismos de segurança e declarações usadas por aplicações sob esta especificação não deveriam impor uma política de segurança particular, porém facilitar a implementação de políticas de segurança específicas para instalações ou aplicações J2EE particulares;

Abstração: Os requisitos de segurança de um componente de aplicação serão logicamente especificados utilizando descritores de distribuição;

Independência: Comportamentos de segurança requeridos e contratos de distribuição deverão ser implementados utilizando uma variedade de tecnologias de segurança populares;

Teste de Compatibilidade: Os requisitos de segurança da arquitetura J2EE deverão ser expressos de uma maneira não ambígua e que permita a determinação de quando a implementação é ou não compatível;

Interoperabilidade Segura: Componentes de Aplicação executando em um produto J2EE devem ser capazes de invocar serviços oferecidos por outros fabricantes J2EE.

2.4 Conclusão

Para que o desenvolvimento deste projeto fosse satisfatório, foi identificada a necessidade do conhecimento de conceitos pertinentes às tecnologias de criptografia, orientação a objetos, linguagem de programação Java e servidores de aplicação Web.

3 Capítulo - Infra-estrutura de um servidor de aplicação Web

3.1 Visão Geral de um Servidor de Aplicação

Um servidor de aplicação provê um ambiente aberto para computação distribuída. Usuários e processos em uma variedade de plataformas podem interagir utilizando as facilidades oferecidas pelo servidor de aplicação.

3.2 Computação Cliente / Servidor em três camadas

A configuração comum de sistemas cliente/servidor utiliza três camadas: um cliente que interage com o usuário, um servidor de aplicação que contém a lógica de negócios da aplicação e um gerenciador de recursos que armazena os dados. Neste modelo o cliente não precisa ter prévios conhecimentos sobre o gerenciador de recursos. Se for alterado o banco de dados utilizado, o servidor deverá ser modificado, mas o cliente não. Em decorrência do fato de que geralmente existem poucas cópias do servidor e estas estão em localizações onde podem-se realizar atualizações facilmente, o processo de atualização é simplificado.

3.3 Os componentes do ambiente de um Servidor de Aplicação

Aplicações baseadas em navegador: Permitem aos usuários enviar e receber informações de servidores Web utilizando o protocolo HTTP. Existem vários tipos de

aplicações baseadas em navegador: *Applets Java*, *Servlets Java* e *Java Server Pages* (JSP).

Servidores Web : Aplicações baseadas em navegador necessitam que um servidor Web seja instalado em pelo menos uma máquina do ambiente do servidor de aplicação.

Servidores de Aplicação e *Enterprise Beans*: O servidor de aplicação contém um ou vários *enterprise beans*, que encapsulam a lógica de negócios e dados utilizados e compartilhados por aplicações EJB. Os *enterprise beans* são instalados em um servidor de aplicação e não se comunicam diretamente com o servidor. Um *container* EJB provê a interface entre o *enterprise bean* e o servidor de aplicação, provendo muitos serviços de baixo nível como gerenciamento de *threads*, suporte a transações e gerenciamento do armazenamento de dados.

Aplicações Java : Aplicações Java podem interagir diretamente com servidores de aplicação utilizando *Java Remote Method Invocation* sobre o protocolo Internet Inter-ORB (RMI/IIOP).

Fontes de Dados: Os *Session Beans* encapsulam por pequenos espaços de tempo tarefas e objetos específicos de clientes e os *Entity Beans* que encapsulam dados permanentes e persistentes. O servidor de aplicação armazena e retém os dados persistentes no banco de dados.

3.3.1 *Applets* e *Servlets Java*

Applets Java são aplicações executadas em um navegador que estende suas capacidades. *Applets* podem ser desenvolvidos utilizando pacotes padrões encontrados nas distribuições Java ou utilizando componentes de *Java Foundation Classes* (JFC).

Para um *Applet* ser executado em um navegador, este deve possuir suporte às classes utilizadas dentro do *applet*; contudo, muitos navegadores podem ser atualizados para suportar o último SDK instalando-se os *plug-ins* necessários.

Servlets Java são executados em servidores Web habilitados para Java e estendem as capacidades do servidor. *Servlets* são programas Java que utilizam a API, classes e métodos associados ao *Java Servlet*. Em adição à API *Java Servlet*, *servlets* podem utilizar pacotes Java que estendem e adicionam características à API. *Applets* podem ser projetados para interagir com *servlets*, mas isto não é necessário.

Servlets estendem as capacidades de servidores Web, criando um sistema para prover serviços de requisições e respostas na Web. Quando clientes enviam uma requisição ao servidor, este pode enviar a informação da requisição para um *servlet* que então monta a resposta que o servidor envia novamente ao cliente.

Diferente dos programas *Common Gateway Interface* (CGI), que requerem um processo inteiro para servir uma requisição do cliente, *servlets* podem atender requisições utilizando *threads*. Esta capacidade torna o *servlet* muito mais eficiente que programas CGI.

Um *servlet* pode ser carregado automaticamente quando o servidor Web é inicializado, ou pode ser carregado na primeira vez que um cliente requisita este

serviço. Após carregado, um *servlet* continua sendo executado, aguardando requisições adicionais de clientes.

Servlets realizam uma grande variedade de funções. Por exemplo, um *servlet* pode:

- Criar e retornar uma página HTML inteira com conteúdo dinâmico baseado na natureza da requisição do cliente;
- Criar uma porção de uma página HTML (um fragmento) que pode ser inserido em uma página HTML;
- Comunicar com outros recursos do servidor, incluindo banco de dados e aplicações baseadas em Java;
- Servir várias conexões, aceitando requisições e enviando respostas a múltiplos clientes;
- Abrir uma nova conexão no servidor para um *applet* e mantê-la aberta, permitindo várias transferências de dados em uma única conexão. O *applet* pode também iniciar uma conexão entre o navegador cliente e o servidor, permitindo ao cliente e servidor facilmente realizar a "conversação". A comunicação pode ser realizada através de um protocolo customizado ou de um protocolo padrão, como IIOP;
- Filtrar dados através do tipo MIME para processamentos especiais, como conversão de imagens ou *server-side includes* (SSI);
- Prover processamento customizado para qualquer rotina padrão. Por exemplo, um *servlet* pode modificar como um usuário é autenticado.

3.3.2 *Java Server Pages*

Arquivos JSP são similares em alguns aspectos a SSI em páginas HTML estáticas porque ambos inserem funcionalidades de *servlet* em páginas Web. Contudo, em um SSI, uma chamada a um *servlet* é inserida com *tags* especiais de *servlet*. No JSP, o código *servlet Java* é inserido diretamente na página HTML.

Uma das várias vantagens do JSP é que ele permite efetivamente separar código HTML de sua lógica de negócios em suas páginas HTML. Pode-se utilizar JSP para acessar componente re-usáveis, como *servlets*, *Java beans*, *enterprise beans* e aplicações Web baseadas em Java.

3.3.3 Servidores Web

O servidor Web provê a comunicação entre a aplicação baseada em navegador e os outros componentes do servidor de aplicação. O servidor de aplicação geralmente possui um *servlet engine*, responsável pelo processamento dos *servlets*, que é independente do servidor Web e do sistema operacional no qual é executado.

3.3.4 Servidores de Aplicação e *Enterprise Beans*

Enterprise Bean é um componente Java que pode ser combinado com outros *enterprise beans* e outros componentes Java para criar uma aplicação distribuída de três camadas. Um servidor de aplicação provê o ambiente de execução para *enterprise beans*, servindo tarefas de baixo nível como gerenciamento de transações e segurança.

Existem dois tipos de enterprise beans:

Um *entity bean* encapsula dados permanentes, que são armazenados em fontes de dados como banco de dados ou sistema de arquivos, e métodos associados que manipulam estes dados. Em vários casos, um *entity bean* deve ser acessado de maneira transacional. Instâncias de *entity bean* são únicas e podem ser acessadas por vários usuários.

Por exemplo, a informação sobre uma conta bancária pode ser encapsulada em uma instância de um *entity bean.* Um *enterprise bean* "conta" deve conter um ID da conta, um tipo de conta e um balanço.

Um *session bean* encapsula uma ou mais tarefas de negócios e dados não permanentes associados com um cliente particular. Diferente dos dados de um *entity bean*, os dados em um *session bean* não são armazenados em uma fonte de dados permanente e não causam danos se estes dados forem perdidos. Todavia, um *session bean* pode atualizar dados em um banco de dados, geralmente acessando um *entity bean.* Por esta razão, um *session bean* pode ser transacional. Quando criadas, instâncias de *session bean* são idênticas, apesar de que certos *session beans* possam armazenar dados semi-permanentes que tornam-se únicos em certos pontos de seus ciclos de vida. Um *session bean* é sempre associado a um único cliente.

Por exemplo, a tarefa associada com a transferência de fundos entre duas contas bancárias pode ser encapsulada em um *session bean.* Um *enterprise bean* "transferência" pode achar duas instâncias de um *enterprise bean* "conta" (utilizando os IDs das contas), e então subtrair um montante específico de uma conta e adicionar o mesmo montante à outra conta.

Antes que um *enterprise bean* possa ser instalado em um servidor de aplicação, este deve sofrer o processo de *deploy*. Durante este processo, várias classes específicas da aplicação servidor são geradas. O descritor do *deploy* contém atributos e características de ambiente que definem como um servidor de aplicação invocará as funcionalidades de um *enterprise bean*. Todo *enterprise bean* deve possuir um descritor que contém características utilizadas pelo servidor de aplicação, que podem muitas vezes ser configurados para todo o *enterprise bean* ou para métodos individuais do mesmo.

3.4 Visão Geral de Segurança em um Servidor de Aplicação

O modelo de segurança busca:

- Prover um modelo de segurança unificado para ambos os recursos Web e *enterprise beans*. Este modelo de segurança permite que uma única política governe a segurança de páginas Web, *servlets* e *enterprise beans*;

- Gerenciar as políticas de segurança e serviços providos pelo servidor de aplicação de maneira distribuída, consistente e com facilidade de gerenciamento;

- Prover suporte melhorado a:

  - Integração de segurança de arquivos JSP, HTML, servlets e EJBs;
  - Suporte a modos de delegação;
  - Suporte a clientes Java seguros;
  - Suporte melhorado ao diretório LDAP.

Uma aplicação segura representa uma coleção de um servidor seguro e componentes relacionados à segurança que residem em um servidor. Resumindo, a

segurança é um esforço entre a aplicação de segurança e o suporte de segurança encontrado nos servidores. Os servidores de aplicação e Web interagem com a aplicação de segurança provendo suporte à segurança.

O suporte à segurança consiste em dois componentes principais :

- *Plug-in* de segurança: Anexado ao servidor Web. O *plug-in* ajuda na tomada de decisões quando usuários requisitam recursos Web (como arquivos HTML ou *servlets*) a partir de clientes Web (sobre HTTP);
- Colaborador de segurança: Anexado a qualquer servidor de aplicação. O colaborador toma decisões de segurança sobre chamadas de métodos em recursos hospedados nos servidores de aplicação.

Estes dois componentes de execução colaboram com o servidor de segurança presente dentro da aplicação de segurança, na tomada das decisões de autenticação, autorização e delegação.

3.5 Garantindo a segurança de Enterprise Beans

Quando um cliente IIOP tenta executar um método em um *enterprise bean* ou seu *Home*, o servidor de aplicação deve determinar se é permitido ao cliente a realização do método.

Os seguintes passos são executados para determinar se o acesso é permitido :

- O servidor de aplicação identifica o principal (cliente) que está invocando o método. Se o principal não pode ser determinado, a requisição é rejeitada;

- O suporte de execução à segurança determina o conjunto de permissões necessárias para invocar o método do *bean*;
- Se o principal possui pelo menos uma das permissões necessárias, o suporte à segurança habilita a invocação do método;
- Após a realização da checagem de autorização, o suporte à segurança consulta a política de segurança para determinar a identidade que deve ser utilizada para que o método possa ser executado. Qualquer chamada subseqüente ao método será invocada utilizando esta identidade.

3.6 Garantindo a segurança de Recursos Web

Quando um usuário em um navegador requisita um recurso Web (um arquivo HTML ou um CGI-BIN), o servidor Web deve determinar se o recurso está protegido. O servidor Web toma a decisão consultando a política de segurança.

3.7 Elementos de segurança

Os componentes de execução de segurança consultam o servidor de segurança, que controla as políticas de segurança e realizam os métodos de autenticação e autorização. Os componentes de execução de segurança anexados ao servidor Web (*plug-in* de segurança) forçam as políticas de segurança baseadas nas políticas configuradas no servidor de segurança.

3.7.1 Servidor de Segurança

O servidor de segurança, que faz parte da aplicação de segurança, possui essencialmente dois propósitos:

- Centralizar o controle sobre as políticas de segurança (por exemplo, delegação de permissões); e
- Prover serviços centrais de segurança (por exemplo, autenticação e autorização).

Em ambos, é confiado ao servidor de segurança o controle e políticas de segurança. O servidor de segurança suplementa os componentes de execução de segurança, *plug-in* de segurança e colaborador de segurança, executados em cada servidor de aplicação. Para ser mais específico, o servidor Web e o servidor de aplicação acionam o servidor de segurança para prover:

- Autenticação, autorização e delegação de políticas;
- Serviços de autenticação e autorização (incluindo o modelo de autenticação LTPA).

Os componentes de execução de segurança (*plug-in* e colaborador) adquirem informações das políticas de segurança da aplicação de segurança. Como a aplicação de segurança é unida às facilidades de administração do sistema, todas as informações de configuração residem em bancos de dados persistentes associados às facilidades de gerenciamento do sistema. O *plug-in* utiliza políticas de segurança para determinar

quais serviços de autenticação e autorização serão invocados. O colaborador de segurança utiliza estas políticas para tomar as decisões de autenticação e autorização.

3.7.2 Plug-in de Segurança

Quando um usuário tenta acessar recursos Web incluindo páginas HTML estáticas, *servlets* e arquivos JSP, o *plug-in* realiza checagens iniciais de segurança. O *plug-in* protege o caminho da URL, não o caminho físico do arquivo. Desta forma, mesmo se duas URLs apontarem para o mesmo recurso físico, o acesso pode ser permitido a uma URL e negado à outra. Se um recurso Web é protegido utilizando a configuração de segurança, o *plug-in* consultará a aplicação de segurança para tomar as decisões de autenticação e autorização.

3.7.3 Colaborador de Segurança

O colaborador de segurança é componente de todos os servidores de aplicação que interagem com servidores de segurança. Para toda a invocação de métodos remotos de um *servlet* ou *enterprise bean*, o colaborador de segurança:

- Realiza checagem de autorização;
- Grava todas as ações realizadas referentes à segurança;
- Força as políticas de segurança.

3.8 Ambiente Operacional

Na figura abaixo pode-se observar as camadas que compõem o servidor:

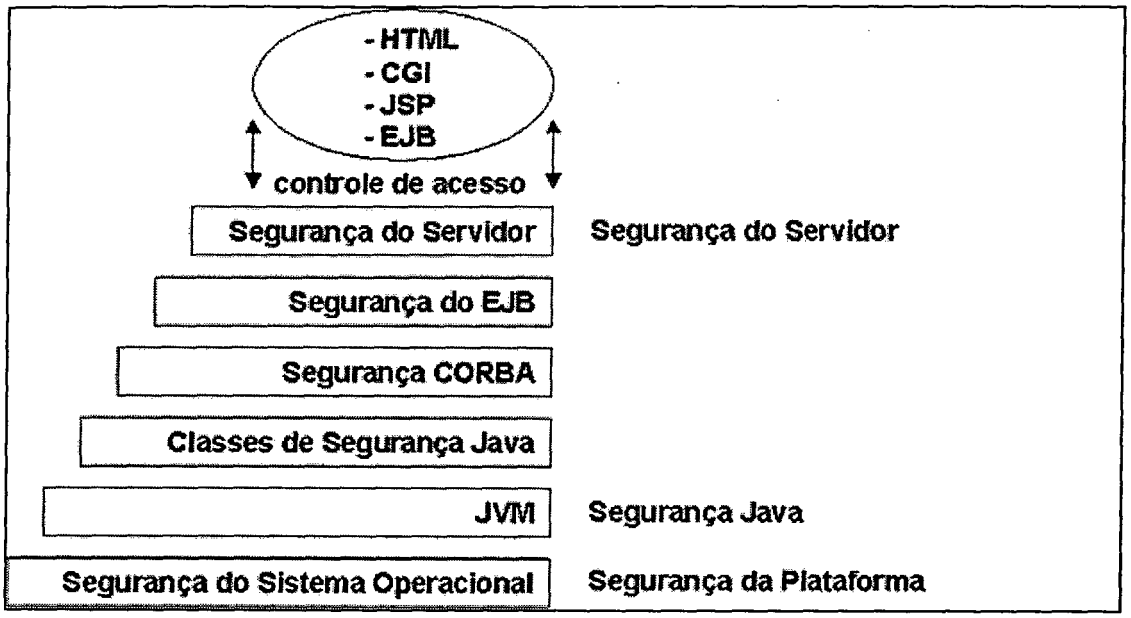

FIGURA 4 – CAMADAS DE SEGURANÇA DO SERVIDOR DE APLICAÇÃO

- Segurança do Sistema Operacional: A infra-estrutura do sistema operacional provê certos serviços de segurança ao servidor de aplicação. O administrador do sistema pode configurar o produto para obter informações de autenticação diretamente do registro do sistema operacional, como o NT *Security Access Manager* – SAM;
- *Java Virtual Machine*: O modelo de segurança da JVM provê uma camada de segurança sobre a camada do sistema operacional;
- Segurança CORBA: Qualquer chamada feita em ORBs seguros é invocada sobre uma camada *Secure Association Service* (SAS) que configura o contexto de segurança e a qualidade necessária de proteção. Após o

estabelecimento da sessão, a chamada é passada para a camada do *enterprise bean*;

- Segurança do EJB: O colaborador de segurança força a segurança do EJB comunicando com o servidor de segurança para serviços de autenticação e autorização;
- Segurança do Servidor de Aplicação: Força políticas de segurança e serviços de maneira unificada nos acessos a recursos Web e *enterprise beans.*

3.9 Modelo de Segurança

3.9.1 Modelo de Autenticação

A autenticação é baseada no tipo de autenticação e no registro do usuário conforme demonstrado na tabela 2. Nos casos onde o ID do usuário e senha são informados para autenticação, esta é delegada ao registro do usuário. Nos casos onde um certificado digital é utilizado para autenticação, as credenciais do certificado são mapeadas para uma entrada associada ao registro do usuário.

	Unix	Windows NT	LDAP
Sistema Operacional Nativo	Utiliza o crypt do sistema, a senha fornecida é criptografada e comparada com a senha do sistema	Autenticação é delegada aos SAM realizando chamadas de sistema para validação de usuário e senha	Não disponível
Sistema Operacional Nativo (Certificados	Não disponível	Não disponível	Não disponível

Digitais)			
LTPA (Usuário e senha)	Não disponível	Não disponível	Uma verificação LDAP é realizada utilizando-se o DN e a senha
LTPA (Certificado Digital)	Não disponível	Não disponível	Baseada na confiança do servidor Web, certificados são validados no estabelecimento de conexões SSL mútuas.

TABELA 2 - MODELOS DE AUTENTICAÇÃO

Se o mecanismo de autenticação é LTPA então a autenticação é delegada a um servidor externo. Se um *token* LTPA que possa ser delegado está disponível, a autenticação é realizada validando o *token*. Em casos onde somente o ID do usuário e senha estão disponíveis, então eles são checados junto ao diretório LDAP.

Se os dados de autenticação são certificados digitais e se o usuário consegue estabelecer uma conexão SSL entre o cliente e o servidor Web, o servidor de aplicação acredita que o certificado pertence ao usuário. A informação do usuário presente no certificado (por exemplo o DN) é então mapeada no registro LDAP do usuário.

3.9.2 Modelo de Capacidade

O modelo de autorização é baseado no clássico modelo de capacidade, no qual as permissões necessárias para realizar uma operação estão associadas ao principal. No caso do modelo de controle de acesso, principais e as operações que eles podem realizar estão associadas aos próprios recursos.

	/hello.html	/servlet/account	/account.html
Alice	HTTP_GET		
Bob	HTTP_GET, HTTP_PUT	HTTP_GET	HTTP_GET
Charlie		HTTP_GET	HTTP_GET

TABELA 3 - MATRIZ DE PROTEÇÃO

Considere-se uma política de autorização que especifique uma matriz de proteção descrita na tabela acima.

A lista de controle de acesso é uma visão baseada em colunas da matriz de proteção. Ela especifica que os recursos hello.html podem ser acessados por Alice e Bob. Já o modelo de capacidade é baseado nas linhas. Este modelo especifica que a usuária Alice tem a permissão para realizar o método HTTP_GET no recurso /hello.html.

No modelo de capacidade a permissão é baseada na operação de requisição e no recurso requisitado. Para facilitar a administração, recursos são agrupados em aplicações e métodos são agrupados em grupos de métodos. O par {aplicação, grupo do método} constitui as permissões necessárias para invocar um método em um recurso.

O administrador :

- Mapeia recursos em aplicações;
- Mapeia métodos dos recursos em grupos de métodos;
- Garante as permissões aos principais.

Quando um usuário (principal) necessita realizar uma operação em um recurso, o suporte à segurança considera um conjunto de permissões para realizar a operação sobre os recursos. Se o principal tem garantido pelo menos umas das permissões necessárias, então o subsistema de segurança autoriza a requisição a ser processada.

Considere-se que hello.html pertence a uma aplicação HelloApp e o método HTTP_GET faz parte do grupo de métodos *ReadMethods*. Também, /servlet/account e /account.html pertencem à aplicação *AccountApp*. Realizada esta configuração e as permissões especificadas na tabela anterior, a tabela abaixo ilustra como as permissões são armazenadas:

Principal	**Permissões**
Alice	{ (HelloApp, ReadMethods) }
Bob	{ (HelloApp, ReadMethods), (HelloApp, WriteMethods), (AccountApp, ReadMethods) }
Charlie	{ (AccountApp, ReadMethods) }

TABELA 4 - ARMAZENAMENTO DE PERMISSÕES

3.9.3 Modelo de Delegação

A delegação é realizada quando um cliente utiliza um intermediário para invocar um método num recurso alvo. Dependendo da política de delegação ativa, o intermediário (como um servidor) invoca um método sob certa identidade :

- Identidade do Cliente: A identidade do cliente requisitando a invocação do método;

- Identidade do Sistema: A identidade do sistema que hospeda o recurso intermediário que invocará o método no recurso alvo;
- Identidade especificada: Uma identidade diferente, especificada explicitamente na política de delegação.

Delegação é o processo de encaminhar as credenciais de um principal através das requisições subseqüentes que ocorrem no contexto de trabalho que a originou.

O administrador do sistema configura o modo "Run As" para cada *enterprise bean*. Considere o exemplo onde o recurso A invoca um método no recurso B que então invoca um método no recurso C identificado na fig. 5. Usando a identidade do cliente, recurso B utiliza o ID "usuário A" para invocar o método no recurso C.

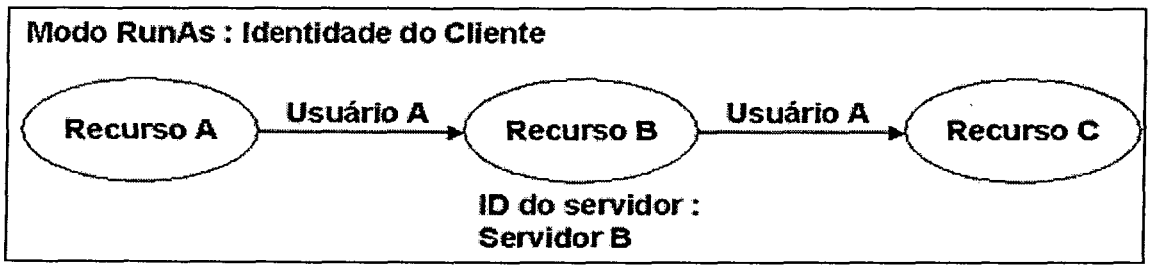

FIGURA 5 - IDENTIDADE DO CLIENTE

Conforme ilustrado na fig. 6 utilizando a identidade do sistema, o recurso B utiliza o ID "servidor B" para invocar o método no recurso C. "Servidor B" é o ID do servidor onde o recurso B reside:

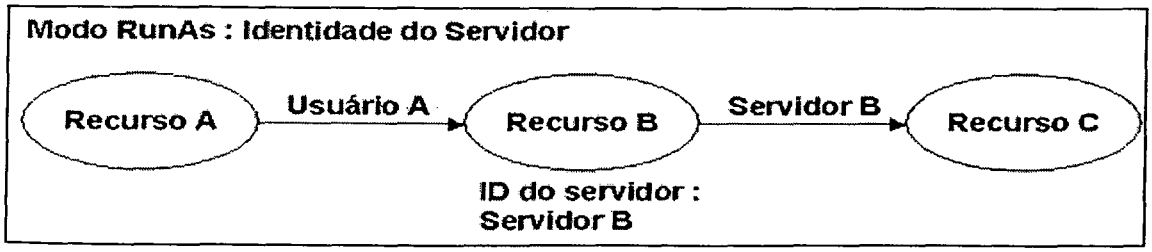

FIGURA 6 - IDENTIDADE DO SERVIDOR

Na fig. 7 utilizando a identidade do cliente, o recurso B utiliza um ID "usuário D" para invocar um método no recurso C. O administrador do sistema configura métodos no recurso B para o "usuário D" como identidade especificada.

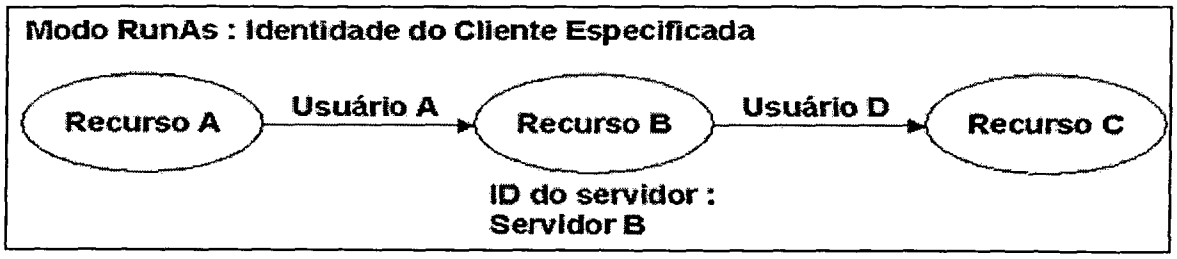

FIGURA 7 - IDENTIDADE DO CLIENTE ESPECIFICADA

3.10 Configuração de Políticas de Segurança

3.10.1 Política de Autenticação

Autenticação é o processo que verifica se um usuário é quem ele diz ser. Autenticação é geralmente realizada em dois passos:

- Adquirir os dados de autenticação de um principal;
- Validar os dados de autenticação no registro de usuário.

O mecanismo de segurança autentica o principal baseado na política de autenticação relacionado ao recurso que o principal requisitou. Quando um usuário requisita um recurso protegido de um servidor Web ou servidor de aplicação, o servidor autentica o usuário.

É suportado o mecanismo de autenticação baseado na validação das credenciais, como certificados, *tokens* e pares de IDs de usuário e senha.

Também são suportados mecanismos de autenticação externos. O cliente e o servidor são autenticados por um servidor externo LTPA. Qualquer esquema de autenticação deve assumir que o cliente não confia no servidor e vice-versa. O maior benefício desta autenticação é que o registro do usuário pode ser centralmente administrado.

Na política de autenticação para realizar a autenticação entre um usuário e o servidor pode ser especificada:

- Mecanismo de Entrada;
- Canal Seguro;
- Mecanismo de autenticação;
- Registro de Usuário.

3.10.1.1 Mecanismo de Entrada

O mecanismo de entrada especifica como um servidor vai obter os dados de autenticação do usuário. O mecanismo pode ser:

Nenhum: O suporte de execução de segurança não questiona o usuário pelas informações de autenticação. Se o recurso requisitado é protegido, então o usuário não será servido pelo servidor, porque não foram informados dados de autenticação;

- Básico (ID de usuário e senha): É solicitado ao cliente Web a informação de um ID de usuário e senha;

- Certificado (X.509): O servidor deve ser configurado para realizar autenticação mútua sobre SSL. Ao cliente é requerida a apresentação de um certificado para estabelecer a conexão. Este certificado é então mapeado para o registro do usuário;
- Customizado: Este mecanismo é útil quando se deseja utilizar um formulário para obter o ID e senha do usuário. O administrador do sistema especifica a URL para qual a requisição do recurso é redirecionada para ser autenticada.

3.10.1.2 Canal Seguro

Uma sessão SSL é requerida para prover confidencialidade e integridade de dados para a informação que trafega entre o cliente e o servidor.

3.10.1.3 Mecanismo de Autenticação

Um mecanismo de autenticação valida os dados de autenticação junto ao registro associado do usuário. Os seguintes mecanismos de autenticação são suportados:

- *Lightweight Third Party Authentication* (Registro de Usuário LDAP);
- Sistema Operacional Nativo (Registro de usuário NT, AIX ou Solaris).

Com o mecanismo de autenticação LTPA, um servidor externo é utilizado para autenticar o usuário. Neste caso, o diretório LDAP deve ser configurado. A autenticação é procedida da seguinte forma:

- O servidor LTPA realiza uma pesquisa no diretório LDAP procurando a entrada de usuário baseada no ID de usuário informado;
- O servidor LTPA realiza um operação *bind* no registro LDAP utilizando o DN e a senha informados pelo usuário;
- Em resultados bem sucedidos da operação *bind*, um *token* LTPA é designado ao usuário. Requisições subseqüentes deste usuário serão baseadas na validação deste *token*.
- Se o *single sig-on* está habilitado, o *token* LTPA será armazenado como uma *cookie* LTPA no navegador do usuário. Como resultado, requisições subseqüentes do usuário serão autenticados validando o *token* contido na *cookie*.

Utilizando o sistema operacional nativo como mecanismo de autenticação, a autenticação será baseada no registro do usuário do sistema operacional executando rotinas nativas para autenticação dos dados.

No caso do sistema operacional UNIX, a senha provida pelo usuário é encriptada e comparada com as senhas armazenadas no arquivo de senhas (/etc/password). No caso do sistema operacional Windows NT, uma chamada de sistema é feita ao sistema de segurança do Windows NT para verificar a senha de determinado usuário.

Em ambos os casos, a aplicação deve ser executada sob uma identidade privilegiada. Para sistemas Unix, tipicamente é utilizado o usuário "root". No caso do Windows NT, o usuário associado deve possuir o privilégio "Atuar como sistema operacional".

3.10.1.4 Registro do Usuário

O registro do usuário é uma visão do domínio do usuário onde informações do usuário e grupo são armazenadas. Um registro de usuário contém mapeamentos do principal com os elementos do domínio. Este contém informações de autenticação e atributos de privilégios de principais. Todo sistema operacional suporta domínio de usuários.

Ligado ao registro do usuário está o domínio de usuários. Um domínio contém todas as informações sobre contas de usuários, informações de autenticação e atributos de privilégios. Em vários sistemas, o registro de usuário é mapeado para o domínio nativo do sistema operacional ou um domínio de usuários da rede.

Principais

Um principal pode ter um conjunto de atributos associados a ele, incluindo identidade de acesso, identidade de grupo e outros. Um principal pode ter uma variedade de atributos de privilégio para controles de acesso como:

Um nome seguro é uma versão amigável de leitura do nome principal. Um principal pode ter um nome seguro associado a ele. Um nome seguro pode ser alterado sem alterar sua identidade de acesso correspondente. Geralmente, somente usuários finais negociam com seus nomes seguros. As várias identidades associadas com o nome seguro, se este existe, são utilizadas sob a cobertura do sistema de segurança. Por exemplo, um principal pode ter um nome seguro "bobs";

O ID de acesso é a identidade de acesso do principal. Tipicamente, é dependente da máquina, uma palavra não legível.

O ID do grupo reflete as afiliações organizacionais do usuário. A utilização de identidades individuais para o controle de acesso pode ser inconveniente se muitos atributos precisam ser alterados quando usuários entram ou deixam a organização. Quando possível, controles de acesso devem ser baseados na construção de alguns grupos. Muitos registros de usuários suportam esta noção e possuem identidades especiais associadas com identidades de grupo.

3.10.2 Políticas de Autorização

Políticas de autorização correlacionam recursos com aplicações e métodos com grupos de métodos. Principais têm garantidas as permissões resultantes destas associações.

Administradores do sistema atribuem recursos a aplicações:

- Recursos são associados com aplicações *enterprise;*
- Recursos Web são associados com aplicações Web;
- Aplicações Web podem ser associadas com aplicações *enterprise.*

Métodos são associados com Grupos de Métodos. Uma permissão é a união de permissões da aplicação *enterprise* e grupos de métodos. O conjunto de permissões necessárias para invocar um método em um recurso é um conjunto da união das permissões de todos os grupos de métodos a que um método pertence e todas as

aplicações a que o recurso pertence. Principais podem ter uma ou mais permissões necessárias, habilitando a autorização a ser bem sucedida quando eles invocam um método.

3.10.3 Políticas de Delegação

A política de delegação é especificada na política "RunAs". A política *RunAs* consiste em duas partes:

O *RunAsMode* especifica se um método deve executar com a identidade do principal que executa a requisição (Identidade do cliente), principal do sistema (Identidade do sistema) ou um principal especificado (Identidade especificada).

O *RunAsIdentity* especifica a identidade do principal se o *RunAsMode* está configurado para identidade especificada.

Valores iniciais do *RunAsMode* são obtidas a partir do descritor de *deploy* do *enterprise bean*. O descritor especificará o modo para o *enterprise bean*. Ele pode opcionalmente especificar o modo para os métodos do *bean*.

3.11 Conclusão

O estudo realizado neste capítulo teve o objetivo de fundamentar o conhecimento da infra-estrutura de um servidor aplicação, demonstrando detalhadamente todas as camadas e componentes que envolvem esta matéria.

Através do conhecimento desta arquitetura é possível identificar os limites de concentração do estudo que será feito na análise de requisitos da aplicação JSP.

4 Capítulo - Análise de requisitos de uma aplicação JSP

Este capítulo analisa e demonstra os requisitos necessários à configuração e disponibilização de um ambiente Web capaz de realizar o processamento de páginas JSP.

Os seguintes componentes são detalhados:

- Servidor Web
- Servidor JSP
- Navegador Web
- Aplicação JSP

4.1 Servidor Web Apache HTTP Server

O servidor Web Apache vem sendo apontado como o mais popular na Internet desde 1996. Em janeiro de 2002 a pesquisa Netcraft Web Server Survey mostrou que 56% dos *sites* Web na Internet utilizam Apache, tornando-o mais utilizado que todos os demais servidores Web juntos.

O projeto Apache HTTP Server é um esforço para desenvolver e manter um servidor HTTP de código aberto para vários sistemas operacionais modernos de servidores e estações, como Windows NT e Unix. O objetivo deste projeto é prover servidor com segurança, eficiência e extensibilidade para prover serviços HTTP em sincronia com os padrões.

4.2 Servidor de Servlets e JSPs Jakarta Tomcat

Tomcat é um *container* de *servlet* utilizado como Referência Oficial de Implementação para tecnologias *Java Servlet* e *Java Server Pages*. As especificações de *Java Servlet* e *Java Server Pages* são desenvolvidas pela Sun pelo *Java Community Process*.

Existem atualmente duas linhas de desenvolvimento do Tomcat. A linha de desenvolvimento da versão 4.0 implementa um novo *container de servlet* (denominado Catalina) que é baseado em uma arquitetura completamente nova. Esta arquitetura busca flexibilidade e performance. A versão 4.0 implementa as especificações *Servlet* 2.3 e JSP 1.2.

A linha de desenvolvimento da versão 3.0 implementa as especificações *Servlet* 2.2. e JSP 1.1. A versão 3.3, desenvolvida atualmente, provê um modelo de *container* modular e permite que possa ser customizado adicionando ou removendo módulos que controlam o processamento de requisições de *servlets*.

4.3 Navegador Web

O cliente Web deve utilizar um navegador Web para enviar as requisições ao servidor Web. Atualmente todos os melhores navegadores oferecem suporte a Java em sua implementação. O suporte a Java é necessário à execução dos *Java Applets*. A execução de *Java Servlets* e JSPs não exige a instalação de suporte Java aos *navegadores* clientes, pois estes componentes são executados no servidor.

4.4 Roteiro para instalação e configuração de um ambiente Apache e Tomcat

4.4.1 Instalação do servidor Web Apache

O servidor Apache é disponibilizado em forma de arquivos binário e código fonte para as distribuições Unix e Windows NT / 2000 .

Aborda-se a compilação e instalação do código fonte do apache em sistemas Unix. O primeiro passo é a realização do *download* do arquivo de instalação. O *download* pode ser realizado a partir do *site* http.apache.org.

Pré-Requisitos:

- São necessários 12 MBytes de espaço em disco para a criação de arquivos temporários. Após a instalação o espaço em disco é de aproximadamente 3 MB, dependendo da quantidade de módulos instalados;
- Assegurar-se que sua instalação possua um compilador ANSI-C;
- Opcionalmente a instalação pode possuir o interpretador Perl 5;
- Para prover maior flexibilidade a instalação de módulos, é necessária a utilização do suporte a *Dynamic Shared Object.*

O próximo passo é configurar o código fonte para plataformas e necessidades particulares. Existem várias opções disponíveis para a instalação. Para a realização de uma instalação padrão necessita-se apenas executar o comando **./configure**.

Para realizar a construção do pacote execute o comando ***make***.

A próxima etapa é instalar o pacote na localização especificada, executando o comando **make install**.

O último passo é editar o arquivo http.conf. Este é o arquivo de configuração do servidor Web Apache. Neste arquivo pode-se especificar atributos como o número da porta do servidor, o ID de usuário utilizado para execução do servidor, o diretório raiz dos documentos servidos, o *e-mail* do administrador do servidor e outros.

4.4.2 Instalação do servidor *Servlet / JSP Tomcat*

Considere-se a instalação do servidor Tomcat em um sistema Unix, utilizando o servidor Apache como servidor Web. O *download* dos arquivos de instalação pode ser realizado no *site* jakarta.apache.org.

Pré-Requisitos

- Servidor Web Apache superior a versão 1.3.20 em correto funcionamento;
- *Java Development Kit* versão 1.3 ou superior.

O primeiro passo é a configuração das variáveis de ambiente JAVA_HOME e TOMCAT_HOME. A variável JAVA_HOME deve apontar para o diretório de instalação da JDK. A variável TOMCAT_HOME deve apontar para o diretório de instalação do servidor Tomcat.

A inicialização do servidor Tomcat pode ser realizada a partir do diretório bin, executando-se o script startup.sh. A saída na tela é a seguinte :

```
2001-09-01 14:23:30 - Http10Interceptor: Starting on 8080
```

2001-09-01 14:23:30 - Ajp12Interceptor: Starting on 8007

2001-09-01 14:23:30 - Ajp13Interceptor: Starting on 8009

Para testar se a inicialização foi correta pode-se digitar no navegador o endereço http://nome_do_servidor:8080. A página inicial do Tomcat será demonstrada.

Para finalizar o servidor Tomcat, a partir do diretório *bin* execute o *script shutdown.sh.*

4.4.3 Integração dos Servidores Tomcat e Apache

Quando o Tomcat é inicializado, automaticamente gera um arquivo de configuração para o Apache em TOMCAT_HOME/ conf / tomcat-apache.conf. Renomeie o arquivo gerado para tomcat-apache_ok.conf e acrescente ao arquivo APACHE_HOME / conf / httpd.conf à seguinte linha :

Include TOMCAT_HOME/conf/tomcat-apache_ok.conf.

A integração já está realizada. Primeiro deve-se iniciar o servidor Apache e em seguida o servidor Tomcat. A configuração gerada automaticamente pelo Tomcat especifica que páginas Web estáticas serão servidas pelo servidor Apache, por ser mais robusto e designado especialmente para esta função. Já os arquivos *servlet* e JSP terão seus processamentos redirecionados para o servidor Tomcat automaticamente.

4.4.4 Utilizando o *JavaSecurityManager* com o *Tomcat*

A utilização do gerenciador de segurança Java junto ao *Tomcat* previne que o servidor seja atacado por *servlets*, JSPs, *beans* JSP e *tag libraries* do tipo *trojan.*

Se alguém autorizado a publicar páginas JSP em seu site incluir a seguinte linha no JSP: <% System.exit(1); %>, toda a vez que o JSP for executado pelo *Tomcat*, este último será finalizado. A utilização do gerenciador de segurança é mais uma linha de defesa que o administrador do sistema pode utilizar para manter seu servidor seguro e disponível.

As classes de permissão definirão quais permissões poderão ser carregadas pelo *Tomcat.* Existe um grande número de classes de permissões na JDK e pode-se criar suas próprias classes para utilização em suas aplicações Web.

Quando o gerenciador de segurança detecta uma violação na segurança, a JVM alerta o fato com *AccessControlException* ou *SecurityException.*

3.4.4.1. Configuração do Gerenciador de Segurança do *Tomcat* em Ambiente *Unix*

A política de segurança implementada pelo gerenciador de Segurança Java é configurada no arquivo tomcat.policy localizado no diretório conf da instalação do *Tomcat.* O arquivo tomcat.policy substitui qualquer arquivo java.policy do sistema.

Este arquivo pode ser editado manualmente ou utilizando-se a ferramenta policytool disponibilizada com a distribuição Java 1.2. As entradas do arquivo tomcat.policy utilizam o formato padrão do arquivo java.policy:

```
// Exemplo de entrada
```

grant [signedBy <signer> [,codeBase <code source>] { permission <class> [<name> [, <action list>]] ; } ;

As entradas signedBy e codeBase são opcionais quando se atribui permissões.

Abaixo encontra-se o arquivo original tomcat.policy:

```
// Permissions for tomcat.
// javac
grant codeBase "file:${java.home}/../lib/-" {
  permission java.security.AllPermission;
};
// Tomcat gets all permissions
grant codeBase "file:${tomcat.home}/lib/-" {
  permission java.security.AllPermission;
};
grant codeBase "file:${tomcat.home}/classes/-" {
  permission java.security.AllPermission;
};
// Example webapp policy
// By default Tomcat grants read access on webapp dir and read of the
// line.separator, path.separator, and file.separator PropertyPermissions.
// Any permissions you grant here are in addition to the default.
grant codeBase "file:${tomcat.home}/webapps/examples" {
  // Allow the example Web application to read all java properties
  permission java.util.ProperyPermission "*", "read";
};
```

Descomentar a entrada do arquivo server.xml para o *ContextInterceptor* que define a classe denominada *PolicyInterceptor.*

Após a configuração dos arquivos tomcat.policy e server.xml para utilização do *SecurityManager*, o *Tomcat* pode ser iniciado com o gerenciador de segurança utilizando-se a opção "-security" do startup.sh.

4.4.5 Tomcat e SSL

Pode-se configurar o servidor *Tomcat* para utilização direta de SSL, ou indiretamente através do servidor Apache. Será abordada a configuração da primeira opção.

É necessário ativar o seguinte fragmento no arquivo de configuração server.xml do *Tomcat* :

```
<Connector className="org.apache.tomcat.service.PoolTcpConnector">
<Parameter name="handler" value="org.apache.tomcat.service.http.HttpConnectionHandler"/>
<Parameter name="port" value="8443"/>
<Parameter name="socketFactory" value="org.apache.tomcat.net.SSLSocketFactory"/>
<Parameter name="keystore" value="/var/tomcat/conf/keystore" />
<Parameter name="keypass" value="changeit"/>
<Parameter name="clientAuth" value="true"/>
</Connector>
```

Geração do Certificado SSL (RSA) para o *Tomcat*:

- Os arquivos *jar* do pacote de segurança Java JSSE *(Java Secure Socket Extension*) devem estar especificados na variável de ambiente CLASSPATH

- Editar o arquivo JAVA_HOME/jre/lib/security/java.security. Adicione security.provider.2=com.sun.net.ssl.internal.ssl.Provider
- Executar o comando keytool –genkey –alias tomcat –keyalg RSA
- Copiar os arquivos jcert.jar, jnet.jar e jsse.jar no diretório JAVA_HOME/jre/lib/ext .

4.5 Aplicação JSP

Os arquivos CompanyAdd.jsp e CompanyAddProcess.jsp fazem parte de uma aplicação Web para automação de força de vendas. O arquivo CompanyAdd.jsp é o formulário utilizado para cadastro de empresas clientes. Este formulário permite a inclusão de todos os dados pertinentes à empresa, como dados cadastrais de endereço, gerente de contas da empresa, área de atuação etc. O arquivo CompanyAddProcess.jsp realiza o processamento dos dados informados no formulário. Ele verifica se todas as informações obrigatórias foram realizadas e opera a inclusão da empresa cliente no banco de dados.

4.5.1 Ferramentas utilizadas na construção do código JSP

Apresenta-se a seguir três ferramentas que podem ser utilizadas na construção de aplicações Web :

- *Forte for Java*
- *IBM VisualAge for Java*
- *Borland Jbuilder*

4.5.1.1 Forte for Java

A ferramenta *Forte for Java* apresenta ambiente de desenvolvimento integrado com a utilização de módulos customizáveis para o desenvolvimento de aplicações Java. A plataforma oferece ambientes visuais para criação, editoração e *debug* das aplicações.

O *Forte for Java* é distribuído em duas versões:

A versão "Community Edition", gratuita, permite aos desenvolvedores a criação de aplicações isoladas, *Java Applets* e aplicações Web de conteúdo dinâmico. Especificamente esta versão permite o desenvolvimento de *JavaBeans*, JSPs, *Servlets*, JDBC e aplicações *Java Data Objects*.

A versão "Enterprise Edition" permite o desenvolvimento de serviços Web. Esta versão possui, além de todas as funcionalidades da versão anterior, a utilização de módulos, modelos, assistentes e geradores que permitem o desenvolvimento de aplicações J2EE e XML.

4.5.1.2 IBM VisualAge for Java

A versão atual do *VisualAge for Java* é 4.0. Esta versão permite a interoperabilidade com outras ferramentas do mercado como *Rational Rose*, *Merant PVCS* e *Rational ClearCase*.

O ambiente de desenvolvimento do *VisualAge* permite que múltiplos desenvolvedores trabalhem em múltiplos projetos, utilizando o controle automático de versão que permite a criação e desenvolvimento rápidos de aplicações.

A ferramenta de desenvolvimento *VisualAge for Java* possui integração com o servidor de aplicação *IBM WebSphere*. Esta integração facilita o desenvolvimento de

aplicações, pois todos os componentes criados com o *VisualAge* podem ser testados na unidade de testes do *WebSphere*.

4.5.1.3 Borland JBuilder

A ferramenta de desenvolvimento *Jbuilder*, da Borland, apresenta as versões *Personal, Professional, Enterprise* e *Enterprise Studio*.

Como características principais da ferramenta, destacam-se:

- Editor visual para rápido desenvolvimento de EJBs versão 2.0, incluindo descritor de *deploy*;
- Integração com os servidores de aplicação *Borland Enterprise Server, BEA WebLogic, IBM WebSphere* e *iPlanet Application Server;*
- Ferramentas, assistentes e componentes que simplificam a administração e o desenvolvimento do banco de dados;
- Desenvolvimento de aplicações Web, utilizando JSP e *servlets;*
- Visualização de código UML;
- Integração com sistemas de controle de versão;
- Utilização de ferramentas XML para fácil integração com a lógica de negócios.

4.6 Abordagem do problema do JSP não ser compilado

Como pode ser verificado na análise dos arquivos JSP da sessão anterior, estes encontram-se texto plano. Esta é uma característica que torna fácil o desenvolvimento dos arquivos JSP, pois permite a fácil inclusão da lógica de negócios no código html.

Mas esta mesma característica apresenta como ponto negativo a falta de segurança, por apresentar o código sem nenhuma forma de encriptação.

Os *Java servlets* e *Java enterprise beans* são classes compiladas. O arquivo gerado para execução não pode ser lido, a menos que se utilizem ferramentas de engenharia reversa como o JAD. A distribuição destes componentes não apresenta um risco de segurança, por não apresentar os códigos da aplicação.

Diferentes destes componentes, os arquivos JSP são compilados somente após sua primeira execução. A compilação dos arquivos JSP gera classes de Java Servlets. A aplicação é distribuída na forma de arquivos JSP e não *servlets*. A distribuição destes arquivos pode ser vista como uma brecha de segurança, pois os arquivos apresentam-se em texto plano, comprometendo a garantia de propriedade intelectual da aplicação.

4.7 Soluções possíveis

O problema de segurança causado pelo fato do arquivo JSP estar apresentado em texto plano pode ser resolvido utilizando-se o seguinte conjunto de ações:

1. Realizar a pré-compilação do arquivo JSP;
2. Empacotar os recursos Web em arquivos .war;
3. Assinar o arquivo .war para garantir sua autenticidade.

4.8 Realização de pré-compilação do arquivo JSP

Na distribuição de arquivos JSPs é desejável que seus clientes não recebam estes arquivos em texto plano. A distribuição destes arquivos deve ser realizada de maneira já compilada (.class).

Existem ferramentas no mercado que realizam a pré-compilação destes arquivos e os servidores de aplicação podem ser configurados para sempre executar o arquivo já compilado. Além de retirar uma brecha de segurança em sua aplicação, a pré-compilação destes arquivos gera um ganho de performance, pois estes arquivos não necessitam sofrer compilação em sua primeira execução.

Uma ferramenta que realiza esta pré-compilação dos arquivos JSP é o JSPC, distribuído com o servidor de aplicação Jrun. Compiladores semelhantes ao JSPC permitem que todos os arquivos JSP de uma aplicação sejam compilados em um único comando.

4.9 Empacotamento de recursos Web em arquivos war

Clientes Web são empacotados em arquivos de aplicações Web. Em adição a componentes Web, arquivos de aplicações Web podem conter outros arquivos, incluindo os seguintes:

- Classes do servidor, geralmente componentes da arquitetura *JavaBeans;*
- Conteúdo Web estático, como arquivos de imagem e som;
- Classes do cliente.

Um arquivo WAR (*Web Archive*) é um arquivo único que contém todos os componentes de uma aplicação Web e organiza de maneira hierárquica estes componentes. Trata-se de um veículo para distribuir aplicações Web para um ambiente alvo.

Neste ambiente, uma ferramenta irá desempacotar o arquivo WAR e colocar seus arquivos na estrutura especificada pelo arquivo descritor.

Arquivos WAR são identificados pela extensão .war. Podem ser criados por qualquer ferramenta *Java archive* (JAR) . É possível que qualquer arquivo (ou todos) seja incorporado ao arquivo war.

4.10 Assinatura de arquivo war com jarsigner

A ferramenta *jarsigner, JAR Signing and Verification Tool*, gera assinaturas para arquivos JAR e permite realizar a verificação das mesmas.

Uma assinatura digital possui as seguintes características:

- Sua autenticidade pode ser verificada computacionalmente utilizando-se uma chave pública correspondente à chave privada utilizada para criar a assinatura;
- Não pode ser forjada, assumindo-se que a chave privada mantenha-se secreta;
- Os dados assinados não podem ser alterados. Se isto ocorrer a assinatura não valida mais sua autenticidade.

Jarsigner utiliza informações de chaves e certificados de um *keystore* para gerar assinaturas digitais para arquivos JAR. Um *keystore* é um banco de dados de chaves

privadas e os certificados X.509 associados utilizados para autenticar as chaves públicas. A ferramenta *keytool* é utilizada para criar e administrar *keystores.*

O *jarsigner* utiliza uma entidade de chave privada para gerar a assinatura. O JAR assinado contém entre outras coisas uma cópia do certificado do *keystore* para a chave pública correspondente à chave privada utilizada na assinatura do arquivo. O *jarsigner* pode verificar uma assinatura digital de um arquivo JAR utilizando um certificado próprio (inserido num bloco de arquivo da assinatura) .

Até o momento, *jarsigner* pode assinar somente arquivos JAR criados pela ferramenta *Java archive* do JDK. Por este motivo arquivos WAR também podem ser assinados.

O comportamento padrão do comando *jarsigner* é assinar um arquivo JAR. Para verificar a assinatura do arquivo utilize a opção "- verify".

Neste exemplo, deseja-se assinar um arquivo chamado bundle.war utilizando-se a chave privada da usuária "jane", na *keystore* denominada "mystore" localizada no diretório C:\working . A senha da *keystore* é "mypass" e a senha da chave privada de jane é "j638klm". Utilizar o comando abaixo para assinar e renomear o arquivo para "sbundle.war":

```
jarsigner -keystore C:\working\mystore -storepass myspass
-keypass j638klm -signedjar sbundle.war bundle.war jane
```

4.11 Conclusão

A proteção dos arquivos JSP pode ser feita através da utilização de ferramentas de criptografia e compilação de código livre aliadas a utilização de políticas de segurança no servidor de aplicação.

O estudo mostrou que pela complexidade das variáveis envolvidas, podem existir diversas possibilidades de vulnerabilidades nos projetos comerciais em produção.

Observou-se que pode-se combinar tecnologias de criptografia e segurança para aumentar o nível de confiabilidade no desenvolvimento de novas aplicações.

Os resultados obtidos indicam um ponto para revisão de arquitetura de engenharia de *software* e servidores de aplicação disponíveis atualmente.

Pode-se concluir que é possível e viável minimizar o impacto causado por este processo através da convergência das propostas expostas.

5 Capítulo - Conclusão

5.1 Contribuições

O crescente uso da Internet nos negócios entre empresas aumentaram a demanda de troca de informações e colaboração entre suas cadeias de distribuição, aumentando drasticamente a necessidade de garantir a integridade dos dados que circulam na rede.

A economia mundial está caminhando para digitalização em larga escala, e não pode ter sua infra-estrutura baseada em um modelo frágil e não confiável. Qualquer suspeita ou prática efetiva que seja colocada em relação a esta conjuntura pode dirigir rapidamente para o caos todo o sistema.

O presente trabalho propôs através do tema definido "Infra-estrutura de um servidor de aplicação Web seguro", a análise de segurança de um servidor de aplicação JSP e suas possíveis vulnerabilidades. Foi apresentada sugestão para contornar esta deficiência e levantar os requisitos mínimos que a construção de uma aplicação deve respeitar.

De acordo com os testes realizados, o servidor de aplicações apresentou vulnerabilidades, e pode-se dizer que os resultados obtidos foram satisfatórios, tendo em vista que a partir da aplicação da solução o problema não foi mais constatado.

A grande vantagem do modelo proposto reside no fato de não ser necessário rescrever ou analisar os requisitos da aplicação instalada no servidor de aplicação. Ela pontualmente propõe corrigir esta vulnerabilidade através da utilização de uma ferramenta direcionada ao problema especificado.

O padrão Java J2EE garante ao desenvolvedor a criação de uma arquitetura segura para realização de transações na Internet, sendo que a dissertação focou-se no estudo de uma aplicação de 3 camadas Java que é suportada no servidor de aplicação e como as técnicas de criptografia e programação da Linguagem Java podem complementar a segurança desta infra-estrutura.

Foi também elaborada proposta para suprir ou minimizar esta brecha na arquitetura atual e o estudo indica a possibilidade de resolução através da incorporação do método aplicado na infra-estrutura do servidor.

As avaliações quanto ao desenvolvimento foram realizadas com o auxilio da linguagem de modelagem UML e as técnicas de investigação de software de acordo com o próprio padrão especificado pela linguagem JAVA.

5.2 Perspectivas para trabalhos futuros

São várias as possibilidades a serem exploradas no campo de segurança relativas a um servidor de aplicação, pois o ambiente em que está inserido comporta uma infinidade de variáveis que podem ser aprofundadas e discutidas.

Uma possível proposta seria a integração do método apresentado à estrutura nativa de um servidor de aplicação, ou seja, o desenvolvimento integrado junto a um servidor de aplicação de código livre e a sua execução de maneira que seja possível corrigir esta disfunção até hoje apresentada.

Existe ainda a necessidade de avaliar mais detalhadamente os outros componentes que constituem esta arquitetura para que possíveis pontos sejam estressados de maneira a garantir maior confiabilidade em relação ao conjunto.

Um dos pontos que também deve ser considerado está relacionado às técnicas de desenvolvimento de software e a criação de uma arquitetura que seja capaz de prever ou reportar desvios nos comportamentos considerados padrões utilizando os conceitos de inteligência artificial ou simplesmente através do adequado tratamento de exceções.

6 Anexo – Código Fonte Java JSP

CompanyAdd.jsp

```
<html>
 <head>
 <jsp:useBean id="stringBundle" class="edu.udesc.opensales.arch.GetStringBundleBean" scope="session"/>
 <jsp:useBean id="business" class="edu.udesc.opensales.bo.MaintainBusinessBean" scope="session"/>
 <jsp:useBean id="interest" class="edu.udesc.opensales.bo.MaintainInterestsBean" scope="session"/>
 <link rel="stylesheet" href="../opensales.css">
 <title><%= stringBundle.getStringBundle("130",session.getValue("locale").toString()) %></title>
 </head>
 <%
  response.setHeader("Expires", "Tues, 01 Jan 1980 00:00:00 GMT");
  response.setHeader("Cache-Control","no-store"); //HTTP 1.1
  response.setHeader("Pragma","no-cache"); //HTTP 1.0
  response.setDateHeader ("Expires", 0); //prevents caching at the proxy server
  int i=0;
 %>
 <body>
  <form name="frmAddCompany" method="post" action="CompanyAddProcess.jsp">
   <jsp:include page="DefHeader.jsp" flush="true"/>
   <table border="0" width="100%" cellspacing="0" cellpadding="0">
    <tr>
     <td class="pagetitle" width="100%" colspan="4"><b><%=
stringBundle.getStringBundle("130",session.getValue("locale").toString()) %></b></td>
    </tr>
   </table>
   <jsp:include page="MainMenu.jsp" flush="true"/>
   <table border="0" width="100%" cellspacing="0" cellpadding="0">
    <tr>
     <td> </td>
    </tr>
    <tr>
```

```
        <td class="label" align="left" width="15%"><b><%=
 stringBundle.getStringBundle("113",session.getValue("locale").toString()).trim() %></b></td>
        <td width="35%"><input name="inpName" type="text" size="50"></td>
       </tr>
       <tr>
        <td class="label" align="left" width="10%"><b><%=
 stringBundle.getStringBundle("132",session.getValue("locale").toString()).trim() %></b></td>
        <td width="35%"><input name="inpAddress" type="text" size="50"></td>
        <td class="label" align="left" width="10%"><b><%=
 stringBundle.getStringBundle("133",session.getValue("locale").toString()).trim() %></b></td>
        <td width="35%"><input name="inpCity" type="text" size="50"></td>
       <tr>
        <td class="label" align="left" width="10%"><b><%=
 stringBundle.getStringBundle("134",session.getValue("locale").toString()).trim() %></b></td>
        <td width="35%"><input name="inpState" type="text" size="50"></td>
        <td class="label" align="left" width="10%"><b><%=
 stringBundle.getStringBundle("136",session.getValue("locale").toString()).trim() %></b></td>
        <td width="35%"><input name="inpCountry" type="text" size="50"></td>
       </tr>
       <tr>
        <td class="label" align="left" width="10%"><b><%=
 stringBundle.getStringBundle("135",session.getValue("locale").toString()).trim() %></b></td>
        <td width="35%"><input name="inpZipCode" type="text" size="50"></td>
        <td class="label" align="left" width="10%"><b><%=
 stringBundle.getStringBundle("137",session.getValue("locale").toString()).trim() %></b></td>
        <td width="35%"><input name="inpPhone1" type="text" size="50"></td>
       </tr>
       <tr>
        <td class="label" align="left" width="10%"><b><%=
 stringBundle.getStringBundle("138",session.getValue("locale").toString()).trim() %></b></td>
        <td width="35%"><input name="inpPhone2" type="text" size="50"></td>
        <td class="label" align="left" width="10%"><b><%=
 stringBundle.getStringBundle("139",session.getValue("locale").toString()).trim() %></b></td>
        <td width="35%"><input name="inpPhone3" type="text" size="50"></td>
       </tr>
```

```
      <tr>
       <td class="label" align="left" width="10%"><b><%=
stringBundle.getStringBundle("140",session.getValue("locale").toString()).trim() %></b></td>
       <td width="35%"><input name="inpFax" type="text" size="50"></td>
       <td class="label" align="left" width="10%"><b>E-Mail</b></td>
       <td width="35%"><input name="inpEmail" type="text" size="50"></td>
      </tr>
      <tr>
       <td class="label" align="left" width="10%"><b><%=
stringBundle.getStringBundle("131",session.getValue("locale").toString()).trim() %></b></td>
       <td width="35%"><input name="inpWebSite" type="text" size="50"></td>
       <td class="label" align="left" width="10%"><b><%=
stringBundle.getStringBundle("141",session.getValue("locale").toString()).trim() %></b></td>
       <td width="35%"><input name="inpCnpj" type="text" size="50"></td>
      </tr>
      <tr>
       <td class="label" align="left" width="10%"><b><%=
stringBundle.getStringBundle("142",session.getValue("locale").toString()).trim() %></b></td>
       <td width="35%"><input name="inpAccountManager" type="text" size="50"></td>
       <td width="10%"></td>
       <td width="35%"></td>
      </tr>
      <tr>
       <td class="label" align="left" width="10%"><b><%=
stringBundle.getStringBundle("71",session.getValue("locale").toString()).trim() %></b></b></td>
       <td>
        <select name="selBusiness" size="1">
         <option value="" selected></option>
         <% if ( business.openConnection() ) {
           if ( business.getAllBusiness() )
           {
            while( business.getNextBusiness())
            {
             int id          = business.getIDColumn("id");
             String name     = business.getColumn("name");
```

```
            i++;
        %>
            <option value="<%=id%>"><%=name%></option>
        <%
            }
           }
           business.closeConnection();
          } else {
           System.out.println("Could not connect to database!");
          }
        %>
       </select>
      </td>
      <td width="5%"></td>
      <td width="35%"></td>
     </tr>
     <tr>
      <td class="label" align="left" width="10%"><b><%=
stringBundle.getStringBundle("74",session.getValue("locale").toString()).trim() %></b></b></td>
      <td>
       <select name="selInterest" size="1">
        <option value="" selected></option>
        <% if ( interest.openConnection() ) {
          if ( interest.getAllInterests() )
          {
           while( interest.getNextInterest() )
           {
            int id          = interest.getIDColumn("id");
            String name     = interest.getColumn("name");
            i++;
        %>
            <option value="<%=id%>"><%=name%></option>
        <%
            }
           }
```

```
            interest.closeConnection();
           } else {
            System.out.println("Could not connect to database!");
           }
          %>
         </select>
        </td>
        <td width="10%"></td>
        <td width="35%"></td>
       </tr>
       <tr>
        <td> </td>
       </tr>
      </table>
      <table border="0" width="100%" cellspacing="0" cellpadding="0">
       <tr>
        <td colspan="2" align="center" width="25%" class="defFooter1">
         <p align="left">
          <input type="submit" value="<%= stringBundle.getStringBundle("33",session.getValue("locale").toString()).trim() %>">
         </p>
        </td>
        <td colspan="2" align="center" width="25%" class="defFooter1">
         <p align="right">
          <input type="button" value="<%= stringBundle.getStringBundle("53",session.getValue("locale").toString()).trim() %>"
onClick="history.go(-1)">
         </p>
        </td>
       </tr>
      </table>
      <script language="javascript">
       document.frmAddCompany.inpName.focus();
       <% if ( session.getValue("error") == "true" ) { %>
         document.frmAddCompany.txamensagem.value = '<%=
stringBundle.getStringBundle((String)session.getValue("message"),session.getValue("locale").toString()).trim() %>';
       <%
```

```
        session.removeValue("error");
       } else {
        session.putValue("message","143");
      %>
        document.frmAddCompany.txamensagem.value = '<%=
stringBundle.getStringBundle((String)session.getValue("message"),session.getValue("locale").toString()).trim() %>';
      <%
       }
      %>
    </script>
    <jsp:include page="DefFooter.jsp" flush="true"/>
  </form>
 </body>
</html>
```

CompanyAddProcess.jsp

```
<html>
<%@ page language="java" import="edu.udesc.opensales.mss.LoginBean"%>
<jsp:useBean id="maintainCompanies" class="edu.udesc.opensales.bo.MaintainCompaniesBean" scope="session"/>

<body>
<%
 String name       = request.getParameter("inpName");
 String address    = request.getParameter("inpAddress");
 String city       = request.getParameter("inpCity");
 String state      = request.getParameter("inpState");
 String zipCode    = request.getParameter("inpZipCode");
 String country    = request.getParameter("inpCountry");
 String phone1     = request.getParameter("inpPhone1");
 String phone2     = request.getParameter("inpPhone2");
 String phone3     = request.getParameter("inpPhone3");
 String fax        = request.getParameter("inpFax");
 String email      = request.getParameter("inpEmail");
 String webSite    = request.getParameter("inpWebSite");
```

```
  String business      = request.getParameter("selBusiness");
  String accountManager = request.getParameter("inpAccountManager");
  String interest      = request.getParameter("selInterest");
  String cnpj          = request.getParameter("inpCnpj");

  if ( maintainCompanies.openConnection() ){
   if (maintainCompanies.addCompany( name , cnpj, address, city, state, zipCode, country,
                          phone1, phone2, phone3, fax, email, webSite,
                          business, interest, accountManager ) == 0) {

    String message = Integer.toString(maintainCompanies.getTextAreaMessage());
    maintainCompanies.closeConnection();
    session.putValue("message",message);
    session.putValue("error","true");
 %>
  <script language="javascript">
   this.location.href = "CompanyAdd.jsp";
  </script>
 <%
   } else {
 %>
  <script language="javascript">
     this.location.href = "CompanyMaintain.jsp";
  </script>
 <%
   }
  } else {
   System.out.println("Could not connect to database");
  }
 %>

</body>
</html>
```

7 Glossário

API Application Programming Interface – interface utilizada pelos programadores para acesso de classes do sistema.

Algoritmo assimétrico - É um algoritmo de criptografia que usa duas chaves: uma chave pública e uma chave privada, no qual a chave pública pode ser distribuída abertamente enquanto a chave privada é mantida secreta. Os algoritmos assimétricos são capazes de muitas operações, incluindo criptografia, assinaturas digitais e acordo de chave. Também conhecido como algoritmo de chave pública.

Algoritmo Simétrico - Algoritmo de criptografia que usa somente uma chave, tanto para criptografar como para descriptografar. Esta chave deve ser mantida secreta para garantir a confidencialidade da mensagem. Também conhecido como algoritmo de chave secreta.

Autenticação - Verificação reivindicada de uma identidade. O processo de determinar a identidade de um usuário que esteja tentando alcançar um sistema.

Checksum - Um valor calculado a partir de parte de dados que pode ser usado para verificar que o dado não foi alterado.

Controle de acesso - Prevenção e controle do uso não autorizado de um recurso. Tarefas executadas por *hardware*, *software* e controles administrativos para monitorar a operação do sistema, garantindo a integridade dos dados, identificando o usuário, registrando os acessos e as mudanças no sistema e permitindo acesso aos usuários.

Criptografia - Princípios, maneiras e métodos para transformar ininteligíveis informações, e também para restaurar informação criptografada para uma forma inteligível.

DES - Algoritmo de criptografia simétrico com chave de 56 *bits*. Existe também uma variação chamada 3DES ou triplo DES, em que se usa três vezes a chave de 56 *bits*. Mesmo resultando em uma chave de 168 *bits*, um tipo de ataque chamado "meet in the middle" pode quebrar um triplo DES com o mesmo esforço que seria necessário para quebrar um algoritmo de 112 bits.

Diffie-Hellman - Um algoritmo assimétrico que permite acordo de chaves: as duas partes trocam suas chaves públicas e as usam em conjunto com suas chaves privadas para gerar uma terceira chave secreta compartilhada. Um curioso que veja as chaves públicas mas não tenha o acesso à chave confidencial de um ou de outro não consegue descobrir a terceira chave compartilhada.

DSA - Algoritmo de assinatura digital é um algoritmo assimétrico que permite criar assinaturas digitais.

DSS (Digital Signature Standard) - Padrão do governo dos EUA que combina DSA e SHA-1 para especificar um formato para assinatura digital.

MAC (Message Authentication Code) - Código de autenticação de mensagem: um algoritmo que usa um algoritmo seguro *hash* e uma chave secreta para garantir a integridade de uma mensagem. O remetente cria um MAC usando a mensagem e a chave secreta; o receptor verifica a mensagem com a mesma chave secreta. Ninguém que não conheça a chave secreta pode modificar a mensagem, porque não podem produzir um MAC que possa ser verificado.

MD5 - Algoritmo seguro *hash* criado por Ron Rivest.

Public key - Uma chave criptográfica disponível para distribuição sem necessidade de segredo. É o oposto de uma chave privada ou chave secreta.

RSA - Algoritmo assimétrico que permite criptografar dados, criar e verificar assinaturas digitais.

SSL Secure Sockets Layer - Protocolo que possibilita realizar comunicações seguras através de criptografia e autenticação.

X.509 - Padrão que especifica o formato dos certificados digitais, de tal maneira que se possa amarrar firmemente um nome a uma chave pública, permitindo autenticação forte.

8 Referências Bibliográficas

[AKL83] Akl, S. **"Digital Signatures: A tutorial Survey."** Computer, February 1983.

[APA02] APACHE: **HTTP Server Project**. Na Internet. [online] Disponível na Internet via WWW. http://httpd.apache.org/. Arquivo acessado em 6 de janeiro de 2002.

[BALD96] Baldwin, R., and Rivest, R. **The RC5, RC5-CBC, RC5-CBC-Pad, and RC5-CTS algorithms**. RFC 2040, October 1996.

[BEA02] BEA: **Rapidly Connecting with Customers and Business Partners**: Na Internet. [online] Disponível na Internet via WWW. http://www.bea.com/solutions/collaboration/index.shtml . Arquivo acessado em 6 de janeiro de 2002.

[BOOC00] Booch, Grady et ali, **UML, guia do usuário**, tradução Fábio de Freitas da Silva, Editora Campus, 2000

[BOR01] BORLAND: **Borland Jbuilder**. Na Internet. [online] Disponível na Internet via WWW. http://www.borland.com/jbuilder/. Arquivo acessado em 16 de Novembro de 2001.

[CHAP95] Chapman, D., and Zwicky, E. **Building Internet Firewalls**. Sebastopol, CA: O'Reilly, 1995.

[COD01] **Coding the FUTURE**: Na Internet. [online] Disponível na Internet via WWW. http://www.jboss.org/. Arquivo acessado em 8 de novembro de 2001.

[CRYP01] Cryptography Research: **Research information on computer security and cryptographic**. Na Internet. [online] Disponível na Internet via WWW. http://www.cryptography.com/. Arquivo acessado em 8 de novembro de 2001.

[C2IT02] C2IT.com **Security Holes**: http://www.devitry.com/c2it-security.html

. Arquivo acessado em 12 de Dezembro de 2002.

[DENN90] Denning, P. Computers Under: **Attack: Intruders, Worms, And Viruses. Reading**, MA: Addison-Wesley, 1990.

[FOR02] FORTE TOOLS: Na Internet. [online] Disponível na Internet via WWW. http://wwws.sun.com/software/Developer-products/. Arquivo acessado em 5 de fevereiro de 2002.

[DIMOV01] DIMOV, Jordan: **JSP Security**. http://softwaredev.earthweb.eom/java/article/0,,12082_883381,00.html Arquivo a cessado em 10 de Dezembro de 2001.

[DIMOV01] DIMOV, Jordan: **JSP Security**. http://www.jadcentral.com/newscentral/feature.jsp?feature_ID=23. Arquivo acessado em 12 de Dezembro de 2002.

[GARF96] Garfinkel, Simson et Spafford, Gene, **Practical UNIX and Internet Security**, 2nd edition, Sebastopol: O'Reilly & Associates, Inc., 1996

[GAR02] GARTNER: **Gartner Group Interactive**: Na Internet. [online] Disponível na Internet via WWW. www.gartner.com. Arquivo acessado em 10 de Março de 2002.

[GIGA02] GIGA: **Giga Group**: Na Internet. [online] Disponível na Internet via WWW. www.giga.com. Arquivo acessado em 10 de Março de 2002.

[IBM01] IBM: **Alpha Works Emerging Technologies**: Na Internet. [online] Disponível na Internet via WWW. www.alphaworks.ibm.com. Arquivo acessado em 14 de Dezembro de 2001.

[IBM01] IBM: **IBM VisualAge for Java** Na Internet. [online] Disponível na Internet via WWW. http://www-3.ibm.com/software/ad/vajava/. Arquivo acessado em 14 de Novembro de 2001.

[JAVA02] Sun Microsystems: **SUN Java Technology**: Na Internet. [online] Disponível na Internet via WWW. http://wwws.sun.com/software/java/. Arquivo acessado em 23 de janeiro de 2002.

[JON02] Jonas Application Server: **JONAS**: Na Internet. [online] Disponível na Internet via WWW. http://www.evidian.com/jonas. Arquivo acessado em 23 de janeiro de 2002.

[JRUN01] JRUN: **Advanced Configuration Guide**: Na Internet. [online] Disponível na Internet via WWW. http://livedocs.macromedia.com/jrun31docs/. Arquivo acessado em 20 de Dezembro de 2001.

[KOVED01] KOVED, Larry **Security challenges for Enterprise Java in an e-business environment**: Na Internet. [online] Disponível na Internet via WWW. http://www.research.ibm.com/journal/sj/401/koved.pdf. Arquivo acessado em 05 de fevereiro de 2002.

[KAHN96] Kahn, David, **The Codebreakers**, New York: Scribner, 1996

[KNUD00] Knudsen, Jonathan et ali, **Aprendendo Java**, Editora Campus, 2000.

[OBI02] OBI: **OBI Information from Supplyworks**.com/obi. Na Internet. [online] Disponível na Internet via WWW. http://www.supplyworks.com/obi/. Arquivo acessado em 05 de janeiro de 2002.

[NEWT97] Newton, David. E, **Encyclopedia of Cryptology**, Santa Barbara: ABC-CLIO, Inc., 1997

[ORA02] ORACLE: **What's The #1 Open Application Server?** Na Internet. [online] Disponível na Internet via WWW. http://www.oracle.com/features/9iAS/index.html?t1as_one.html . Arquivo acessado em 20 de fevereiro de 2002.

[POTE01] **Potential "JSP" vulnerability**: http://screaming-penguin.com/main.php?storyid=2038. Arquivo acessado em 16 de Dezembro de 2001.

[RATI02] Rational UML: **Information about Rational Rose Unified Modeling Language (UML) CASE Tools**: Na Internet. [online] Disponível na Internet via WWW. http://www.rational.com/. Arquivo acessado em 10 de janeiro de 2002.

[ROMAN99] Roman, Ed. **Mastering Enterprise Java Beans and the Java 2 Enterprise Edition**. Wiley Computer Publishing, 1999.

[SEC01] Security Reports: JSP Warnings http://www.j2.ru/frozenfido/ru.linux/203089efb943f.html. Arquivo acessado em 10 de Dezembro de 2001.

[SCHN96] Schneier, Bruce, **Applied Cryptography**, 2nd edition, EUA: John Wiley & Sons, Inc., 1996

[STA 99] Stallings, William; **Cryptography & Network Security**: Principles & Practice, Prentice Hall

[SUN01] Sun Microsystems: **Java Chronology**: Na Internet. [online] Disponível na Internet via WWW. http://java.sun.com/sfaq/chronology.html . Arquivo acessado em 10 de Dezembro de 2001.

[SUN98] Sun Microsystems, **Implementing Java Security**, Sun Microsystems, Inc., Sun Educational Services, 1998

[TANE96] Tanenbaum, Andrew S., **Computer Networks**, 3rd edition, Upper Saddle River: Prentice Hall Ptr, 1996