

SUMÁRIO

LISTA DE ILUSTRAÇÕES	08
LISTA DE TABELAS	09
LISTA DE ABREVIATURAS OU LISTA DE SIGLAS.....	10
RESUMO.....	11
ABSTRACT.....	12
1. INTRODUÇÃO	13
1.1. OBJETIVOS DO TRABALHO.....	16
1.2. ORGANIZAÇÃO DO TRABALHO.....	17
2. REVISÃO BIBLIOGRÁFICA.....	18
2.1. JAVA 2 PLATAFORM ENTERPRISE EDITION – J2EE.....	18
2.1.1. A ARQUITETURA J2EE.....	19
2.2. ARQUITETURA EJB.....	20
2.3. SESSION BEANS X ENTITY BEANS.....	21
2.4. SERVIÇO DISTRIBUÍDO E REMOTE METHOD INVOCATION – RMI.....	23
2.5. OPEN SOURCE.....	24
2.5.1. LIBRARY GENERAL POLICY LICENSE - LGPL	26
2.6. EXTENSIBLE MARKUP LANGUAGE – XML.....	27
2.7. STUB – RPC.....	30
3. JBOSS APPLICATION SERVER.....	32
3.1. INTRODUÇÃO.....	32
3.2 JBOSS CLUSTERING.....	32
3.3. TERMINOLOGIA DE CLUSTERING EM JBOSS.....	33
3.4. CARACTERÍSTICAS DO CLUSTERING JBOSS.....	34
3.5. PARTIÇÕES.....	34

3.6. BALANCEAMENTO DE CARGA.....	35
3.7. JAVA MANAGEMENT EXTENSION – JMX	37
3.8. ARQUITETURA JMX.....	38
3.9. SEGURANÇA EM JBOSS.....	39
3.9.1. J2EE DECLARATIVE SECURITY OVERVIEW.....	40
3.9.2. SECURE REMOTE PASSWORD (SRP) PROTOCOL.....	40
4. INSTALANDO E CONSTRUINDO O SERVIDOR JBOSS	42
4.1. CONDIÇÕES PRÉVIAS PARA INSTALAÇÃO	42
4.2. CONFIGURANDO E INICIANDO O JBOSS.....	45
4.3. ESTRUTURA DE DIRETÓRIO	47
RESULTADOS OBTIDOS.....	51
CONCLUSÃO.....	52
REFERÊNCIAS.....	53
LINKS.....	55

LISTA DE ILUSTRAÇÕES

FIGURA 01 - TELA DE GERENCIAMENTO DE WEB-CONSOLE DO JBOSS.....	14
FIGURA 02 - ARQUITETURA DA PLATAFORMA J2EE	20
FIGURA 03 - CONTAINERS.....	21
FIGURA 04 - PARTIÇÕES.....	35
FIGURA 05 - ARQUITETURA JMX.....	38
FIGURA 06 - EXECUÇÃO DO COMANDO JAVA –VERSION.....	42
FIGURA 07 - EXECUÇÃO DO COMANDO SET.....	43
FIGURA 08 - MODELOS DE ARQUIVOS DE CONFIGURAÇÃO.....	44
FIGURA 09 - TEMPO PARA CARREGAMENTO DO SERVIDOR JBOSS.....	45
FIGURA 10 - TELA DE CONFIGURAÇÃO JMX-CONSOLE DO SERVIDOR JBOSS.....	46
FIGURA 11 - TELA DE CONFIGURAÇÃO DO TOMCAT.....	47
FIGURA 12 - ESTRUTURA DE DIRETÓRIOS NO JBOSS.....	49

LISTA DE TABELAS

TABELA 1 - EMPRESAS QUE UTILIZAM O SERVIDOR DE APLICAÇÕES JBOSS	15
TABELA 2 - DIFERENÇAS BMP-CMP.....	23
TABELA 3 - AMOSTRA PERMISSÃO PARA TEMPO DE MANUTENÇÃO POR PROPORÇÕES DE HA.....	33

LISTA DE ABREVIATURAS E LISTA DE SIGLAS

API – Application Program Interface
CPU - Central Unit of Processing
DTD - Document Type Definition
EJB - Enterprise Java Beans
HTML - HyperText Markup Language
IDE - Integrated Development Environment
IIOP – Internet Inter-Orb Protocol
J2EE – Java Enterprise Edition
JAR - Java Archive
JDBC – Java Database Connectivity (JDBC) API
JDK – Java Development Kit
JMX - Java Management Extension
JNDI – Java Name Directory Interface
JPE - Java Plataform for the Enterprise
JMS – Java Message Service
JSP – Java Server Pages
JTA – Java Transaction API
JTS – Java Transaction Server
JRMP - Java Remote Method Protocol
OSI – Open Source Initiative
RMI - Remote Method Invocation
SO – Sistema Operacional
URL – Uniform Resource Locators
XML – Extensible Markup Language
WWW – World Web Wide

RESUMO

As redes de computadores não emergiram repentinamente como uma tecnologia única e independente, e a mesma deve se aplicar aos sistemas que atravessaram diversas etapas antes de sua aplicação final e continuam em um contínuo ciclo evolutivo conforme as necessidades de mercado e também para acompanhamento de tecnologias que surgem.

Fato este, o crescimento da utilização das redes de computadores, tem levado as empresas a adotar, cada vez mais, novas tecnologias que estão sendo amplamente difundidas e que almejam resultados confiáveis. Neste contexto o Servidor de Aplicações *JBoss* encontra-se em ascensão por ser *Open Source*, disponibiliza diversos recursos onde alguns destes serão abordados neste trabalho tais como: *Clustering*, balanceamento de carga, *JMX*, arquivos básicos de configuração.

Onde reunindo estes recursos em um único produto o *JBoss* tem levado grandes empresas que utilizam ferramentas *WEB* a adotarem este servidor em suas aplicações.

ABSTRACT

The nets of computers didn't emerge suddenly, as an only and independent technology, the same case is applied to the systems that crossed several stages before his final application and they continue in an I click evolutionary I continue according to the market needs and attendance of technologies.

Fact this the growth of the use of the nets of computers, it has been taking the companies to adopt, more and more, new technologies that are being spread thoroughly and that you/they long for reliable results. In this context JBoss Application Server is in ascension for being Open Source, it makes available several resources where some of these will be aborted in this research, fact this gathering their resources has been taking great companies that are used of tools WEB adopt it this servant in their applications. The present work approaches some resources of the application servant such JBoss as: Clustering, load swinging, JMX, basic files of configuration.

1. INTRODUÇÃO

Application Servers, ou servidores de aplicação, são sistemas de *software* que fornecem a infraestrutura de serviços para a execução de aplicações distribuídas. Os servidores de aplicação são executados em máquinas servidoras e são acessados pelos clientes através de uma conexão de rede (LABOUREY; BURKE, 2002).

As vantagens dos servidores de aplicação em relação ao modelo cliente/servidor residem nos serviços implementados por eles e disponíveis, pelo qual as empresas possam concentrar a maior parte do tempo no desenvolvimento da lógica de negócio. Em geral estes serviços diminuem a complexidade do desenvolvimento, controlam o fluxo de dados e gerenciam a segurança.

Um servidor de aplicação é uma plataforma sobre a qual roda a porção servidora de um aplicativo. Isto inclui *hardware* e *software*. O *hardware* está fora do escopo deste trabalho, mas o *software* pode ser dividido em dois grupos: funções do negócio, que são específicas para um domínio de problema; e serviços especializados, que são funções genéricas aplicáveis a diversas soluções.

Assim, do ponto de vista do software um servidor de aplicação consiste de um agrupamento de funções de negócios e de serviços que integrados satisfazem as necessidades dos usuários.

O servidor de aplicação utiliza a arquitetura chamada de 3-camadas ou n-camadas, que permite um melhor aproveitamento das características de cada componente (*servidor de banco de dados, servidor de aplicação e cliente*).

A primeira camada, chamada *Front-End*, usualmente *Browsers*, servem para apresentação e algumas validações. A segunda camada, é a aplicação sendo executada no servidor de aplicação. A terceira camada é o servidor de banco de dados.

Os servidores de aplicação priorizam o compartilhamento de componentes e aplicações, fazendo assim com que seja mais fácil o desenvolvimento, manutenção e gerenciamento de sistemas complexos.

Além das características já citadas, outros serviços também estão disponíveis nos servidores de aplicação:

- *Tolerância à falhas*: através de políticas para recuperação e distribuição de componentes em clones dos servidores;
- *Balanceamento de carga*: a análise da carga nos servidores permite a distribuição de clientes de forma a maximizar a utilização dos recursos disponíveis; (ver capítulo 3, seção 3.6)
- *Gerenciamento dos componentes*: através de ferramentas para a manipulação de componentes e serviços, tais como gerenciamento de sessão, notificação, distribuição da lógica de negócios;
- *Gerenciamento de transações*: garante a integridade da transação em ambientes distribuídos;
- *Console de gerenciamento*: permite o gerenciamento de vários servidores de aplicação através de um único sistema gráfico – conforme figura 1 exemplo de gerenciamento de *web-console* do JBoss;
- *Segurança*: garante a segurança da aplicação. (ver capítulo 3, seção 3.9)

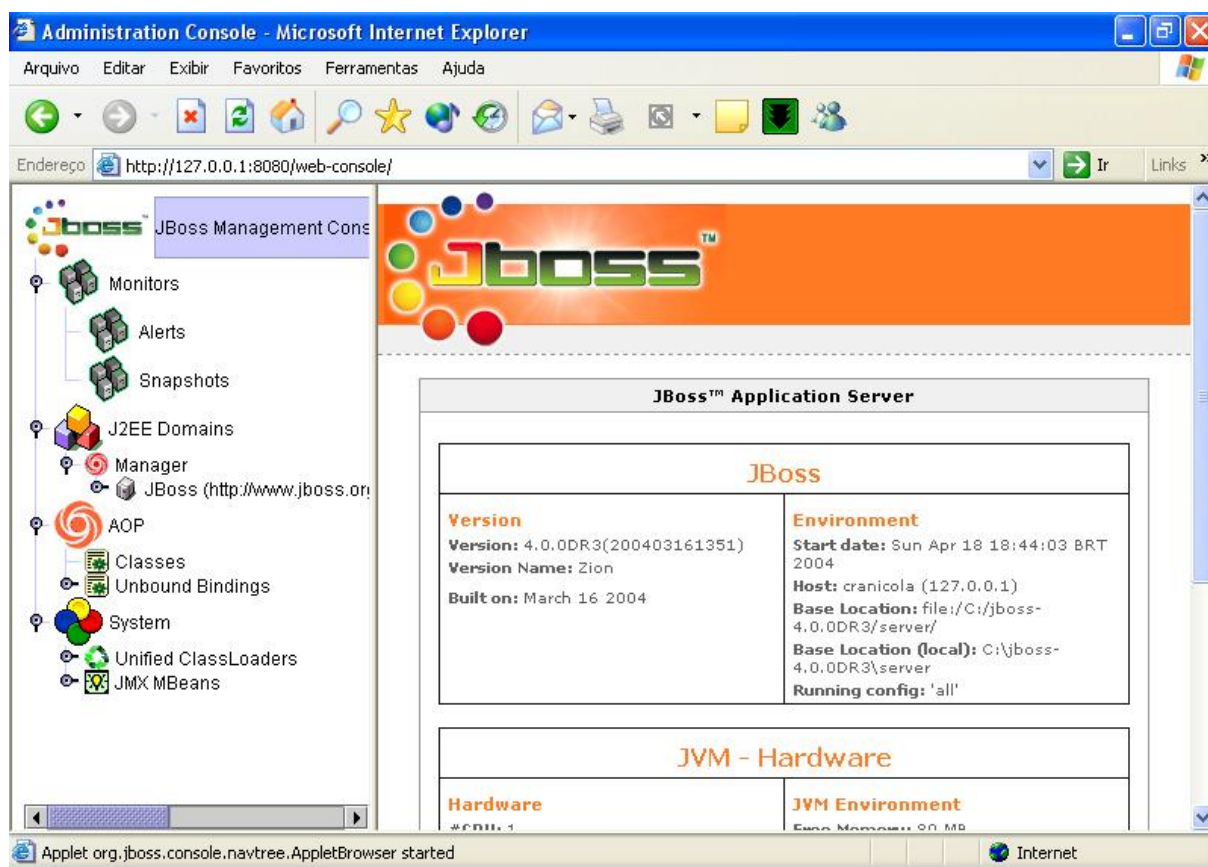


Figura 1 – Tela de gerenciamento de *web-console* do JBoss.

Particularmente, os servidores de aplicação podem ser executados em múltiplos sistemas operacionais, como *Solaris*, *Linux* e *Windows*, o que permite que seja aceitável o desenvolvimento em uma plataforma e sua publicação para produção em outra.

Atualmente com o aumento das aplicações baseadas em ambiente *WEB*, muitos profissionais e empresas vinculadas a esta área buscam soluções que possam agilizar, prover segurança, escalabilidade a cada tarefa a ser executada e onde se pode encaixar o *Jboss* as estas necessidades.

Servidor de Aplicação *JBoss* surgiu destacando-se no mundo *J2EE* (ver capítulo 2, seção 2.1) e *Open Source* (ver capítulo 2, seção 2.5), pelo qual algumas empresas constantes na tabela 1 empregam esta tecnologia:

Accenture	AMD	American Fidelity
Arch Wireless	BASF	BuyMedia
California ISO	Celeris	CTI
Corporate Express	Deloitte & Touche	Dow Jones Indexes
EA Games – Sims Online	Elogex	Ericom Software
FGM	Genscape	Hitachi Data Systems
LastMinuteTravel.com	LeapFrog	Wells Fargo
Lesson Lab	Lion Bio Sciences	McDonalds
McKesson	MCI	Mitre – DISA-DARPA
Motorola	Motability	New York Court Administration
Nextance	Nielsen Media Research	Nortel Networks
Nuasis	Playboy.com	Primus
Sabre – GetThere	Schlumberger	Siemens
QAD	U.S. Department of State	WebMethods

Tabela 1 – Empresa que utilizam o Servidor de Aplicações *JBoss*. – (*JBOSS*, 1999)

Os Sistemas Distribuídos atualmente necessitam, mais do que nunca, dispor das seguintes características:

“Um sistema paralelo ou distribuído que consiste na coleção de computadores interconectados que são utilizados como um só, unificando seus recursos computacionais” (G. Pfister, um dos arquitetos da tecnologia de clusters) (BUNT)

- *Flexibilidade*: Capacidade de poder sofrer mudanças. É evidente que, para isso, a idéia é de que este sistema não perca sua identidade e que estas mudanças não sejam tão complexas de serem realizadas, a ponto de desperdiçar recursos ou até mesmo ser

mais problemática do que a reconstrução do próprio sistema. Ou seja, é muito importante que os sistemas distribuídos, que constantemente enfrentam problemas de complexidade e heterogeneidade, consigam adaptar-se a mudanças facilmente.

- *Interoperabilidade*: Capacidade de interagir com elementos heterogêneos. Um sistema interoperável é um sistema capaz de se comunicar eficientemente, e de forma padronizada com diversas partes. Estas partes podem ser outros sistemas, que por sua vez podem ser sistemas legados, sistemas em linguagens diferentes, etc.
- *Reatividade*: Capacidade de responder rapidamente a mudanças. Com a massificação da associação entre sistemas distribuídos e sistemas de planejamento estratégico e controle de alto risco, cada vez mais se faz necessário que os sistemas distribuídos possuam a característica de reagirem automaticamente a alterações feitas sobre as suas informações de forma a assegurar uma consistência eficiente destas informações.
- *Escalabilidade*: Capacidade de expansão. O quão difícil é determinar o tamanho que o seu sistema tomará no futuro? Como sistemas distribuídos são compostos de elementos, sendo que estes podem variar em quantidade, deve-se capacitar estes sistemas da característica de poder suportar um crescimento. E este crescimento deve ser independente da sua variável sustentadora, como por exemplo, número de usuários, número de máquinas, ou mesmo número de objetos.

1.1 – Objetivos do trabalho

Este trabalho tem como objetivo principal realizar o estudo do servidor de aplicações *JBoss* destacando suas principais características.

Em segundo lugar este trabalho pretende elaborar e disponibilizar um material de consulta que possa ser utilizado por outros profissionais da área interessados no *JBoss*, contendo as suas principais características, tipos e sugestões de instalação.

Finalmente é importante ressaltar que o autor deste trabalho exerce, atualmente, papel de Analista de Sistema na Prefeitura do Município de Paranavaí-Pr. Pretende-se, também, com este trabalho iniciar-se um levantamento e estudo de tecnologias que futuramente poderão ser utilizadas na implementação de sistemas para *WEB* na Prefeitura de Paranavaí.

1.2 – Organização do trabalho

Esta pesquisa é composta de 04 capítulos assim distribuídos:

Capítulo 1 *Introdução* - onde é abordado o que são servidores de aplicação, suas vantagens em relação ao modelo cliente/servidor, suas características, quais tipos de Sistemas Operacionais podem ser utilizados e algumas empresas conceituadas que utilizam o *JBoss*.

No capítulo 2 *Revisão Bibliográfica* - contendo informações básicas sobre as tecnologias adotadas e relacionadas ao *JBoss* como *J2EE*, arquitetura *EJB*, introdução básica do que vem a ser *container*, componente *Beans (Session, Entity)*, introdução ao *JNDI* e *RMI*, conceito de *Open Source*, a licença *LGPL* adotada pelo *JBoss*, *XML* e fechando o capítulo o modelo *RPC*.

No capítulo 3 *JBoss Application Server* - consta uma breve introdução ao *JBoss*, *Clustering* no *JBoss* referindo-se às características, arquitetura do *clustering*, Balanceamento de Carga com uma definição introdutória do que vem a ser o balanceamento de carga, soluções adotando o balanceamento e resultados obtidos com o mesmo, *Java Management Extension – JMX* conceitos básicos e arquitetura, *Segurança em JBoss* com os modelos de segurança.

O capítulo 4 *Instalando e construindo o Servidor JBoss* onde são citados os requisitos básicos necessários para instalação do mesmo, e onde encontrar o *JBoss* e sugestões para configuração, e sua estrutura de diretórios. Finalizando com os capítulos *Resultados Obtidos* descrevendo as facilidades encontradas para instalação do *JBoss* e fechando o trabalho a *Conclusão* final sobre o Servidor de Aplicações *JBoss*.

2. REVISÃO BIBLIOGRÁFICA

Neste capítulo são apresentadas tecnologias e conceitos relacionados ao Servidor de Aplicações *JBoss*.

2.1 – Java 2 Platform Enterprise Edition - J2EE

J2EE é uma tecnologia padrão *Java 2* (SUN, 2004) da SUN. Inclui várias APIs para construção de aplicações *Java* de nível corporativo, incluem *EJB*, *Servlets*, *JDBC*, *JNDI*, *JSP*, *JMS* e transações,

JSP - *JavaServer* é uma tecnologia da Sun que permite misturar conteúdo HTML estático e dinâmico na Web, é um script que roda no lado do servidor, o *JSP* é uma plataforma centrada nos componentes, para que a reutilização de código seja facilitada e para que possam ser criadas aplicações mais poderosas (SUN, 2004).

JTA - *Java Transaction API* é uma especificação de interfaces para o sistema de transações, *JTA* é utilizado por desenvolvedores de *beans* que têm controle explícito (programático) de transações (BMT), suporte por parte do container é obrigatório (SUN, 2004).

JTS - *Java Transaction Service* especifica a implementação de um gerenciador de transação que aceita *JTA* e implementa o mapeamento *Java* da especificação *Object Transaction Service - OTS 1.1 do OMB* no nível abaixo da API (SUN, 2004).

JDBC uma API para conectividade independente do banco de dados entre a plataforma *J2EE* e uma grande variedade de fontes de dados (SUN, 2004).

Enterprise Java Beans são o centro da especificação *J2EE* da Sun. *EJB* são componentes de arquitetura pura do lado-servidor que proporciona suporte embutido para serviços de aplicações como transações, segurança e conectividade de banco de dados.

" Java Beans é um modelo de componentes portátil e independente de plataforma escrito em Java. Ele permite aos desenvolvedores escrever componentes reusáveis e executá-los em qualquer lugar se beneficiando do poder do Java.

Enterprise JavaBeans - EJB é uma arquitetura de componentes multi-plataforma para o desenvolvimento de aplicações Java, distribuídas, escaláveis e orientadas a objetos. EJB torna fácil escrever aplicações de negócios como componentes provendo um conjunto de serviços automáticos para suportar aplicações transacionais." (MUNDO OO, 2004)

2.1.1 A Arquitetura J2EE

Containers e Serviços: A chave da arquitetura *J2EE* é que muito trabalho normalmente feito pelo programador é poupado, já que é feito automaticamente pelo *middleware* - o programador se concentra no *Business Logic*.

A entidade que faz essa mágica é o *Container*, onde um *container* "envolve" um componente de forma a capturar mensagens dirigidas ao componente e fornecer serviços automáticos a este.

Portanto, antes de ser usado, um componente (*seja cliente, Web ou EJB*) deve:

- Ser montado numa aplicação;
- Ser "*deployed*" (*implantado*) dentro de um *container*.

O *container* pode ser configurado em tempo de *deployment*.

- Com *declarative programming*, isto é, mudança de atributos

Exemplos do que se faz no *deployment* ao configurar um *container*:

- Estabelecer segurança
- Estabelecer o tratamento transacional
- Mapear nomes entre a aplicação e os recursos disponíveis

O *container* também gerencia serviços não configuráveis:

- O *lifecycle* dos componentes (*achar, criar, destruir, ...*)
- *Pooling* de recursos (*conexões de bancos de dados, por exemplo*)
- Persistência de dados

Tipos de *Containers*: os seguintes tipos de *containers* existem e executam no servidor *J2EE*:

- *Container EJB*: um tal *container* para acolher algumas ou todas as *Enterprise Beans (EJBs)* de uma aplicação
- *Web container*: um tal *container* para acolher algumas ou todas as *JSPs* e *servlets* de uma aplicação

Os seguintes tipos de *containers* existem e executam na máquina cliente:

- *Application Client Container*: para executar uma aplicação "*console*"

Observe que *servlets* e *JSPs* podem executar sem um "*J2EE server*" completo:

- Podem executar num servidor *Web* com suporte especial sem ter suporte a *EJB*

- Por exemplo: *Apache Tomcat*

Portanto, na figura 1, representa a arquitetura *J2EE*.

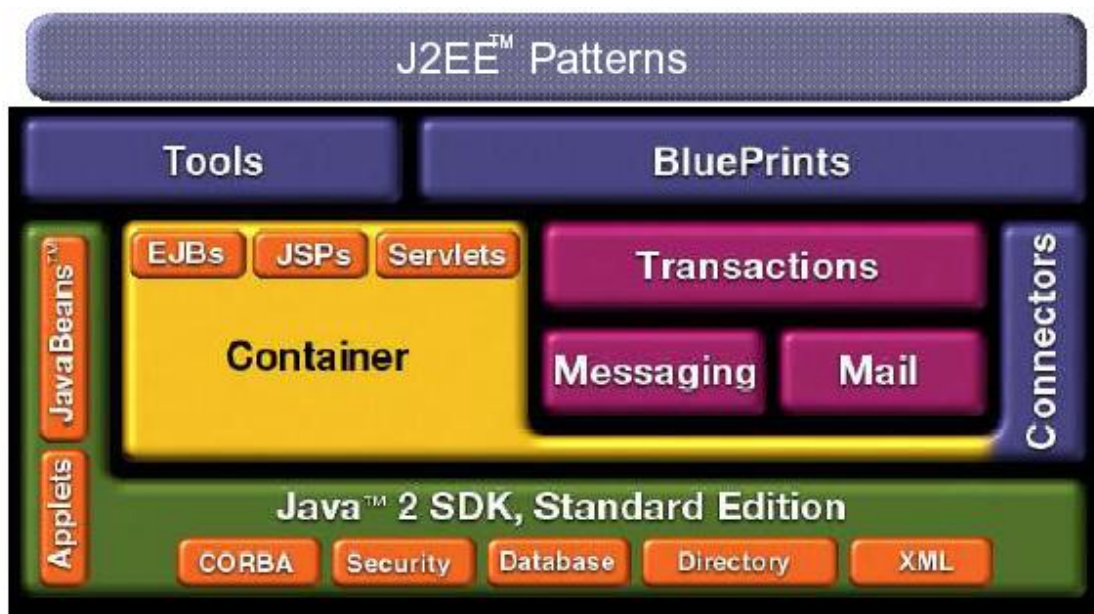


Figura 2 - Arquitetura da plataforma *J2EE* – (ROB JOHNSON, 2003)

2.2 – Arquitetura *EJB*

A arquitetura *EJB* pode ser dividida entre as seguintes funcionalidades: (STARK, 2002)

- § Servidores de Aplicação
- § Containers *EJB*
- § Enterprise Java Beans
- § Clientes *EJB*
- § Sistemas Auxiliares (*J2EE*):
 - *JNDI* (SUN, 2004)
 - *JTS* (SUN, 2004)

EJB provê containers (fornecem suporte em tempo de execução para os componentes *J2EE*) onde os componentes podem ser inseridos no servidor, fornecendo a funcionalidade da aplicação.

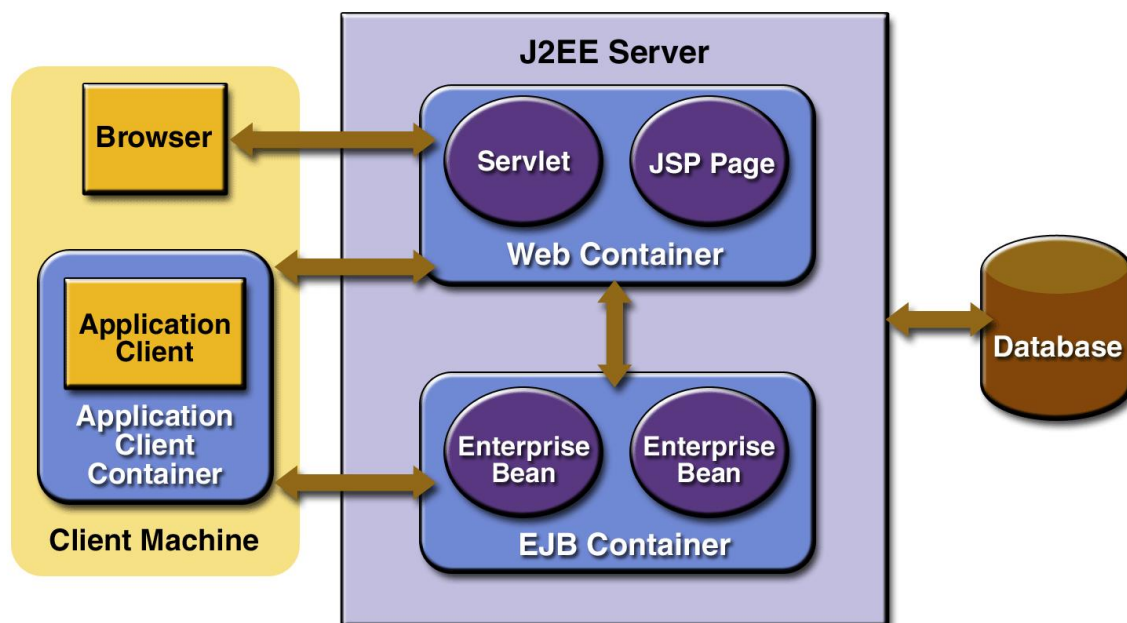


Figura 3 – Container – *J2EE Design and Development* - (ROB JOHNSON, 2003)

Servidor de Aplicações provê *container* para gerenciar a execução de um componente.

O *Container* automatiza as funcionalidades de gerência do ciclo de vida do *EJB*, gerência de estado, segurança, transações distribuídas e persistência dos objetos.

Segundo a SUN (SUN, 2004), a tecnologia do padrão *EJB* permite ao desenvolvedor a independência de plataforma e de fabricante no que se refere ao desenvolvimento de aplicações corporativas multi-camadas. Além disto, a complexidade de desenvolvimento de aplicações distribuídas é consideravelmente simplificada.

2.3 – *Session Beans X Entity Beans*

Segundo a SUN (SUN, 2004) a tecnologia *EJB* define dois tipos de componentes: *Session Beans* e *Entity Beans*

Session Beans

Entender o funcionamento de cada um desses tipos de *EJB* é fundamental para a adequada construção de uma aplicação *J2EE*. Os *Session Beans* são os mais simples, leves e eficientes dos “grãos”. E são ainda classificados como *Stateless*, ou *Stateful*. Um *Stateless Session Bean* não mantém informações sobre o seu contexto de execução. Ele é preparado pelo *container*, executa uma tarefa em favor do cliente e logo em seguida é devolvido para o *pool* de *beans*, mantido pelo servidor de aplicações para maximizar a eficiência.

Já um *Stateful Session Bean* atende, exclusivamente, ao cliente para qual foi criado. Esses componentes são usados para controlar transações que se desenvolvem em várias etapas. O ciclo de vida do *Stateful Session Bean* é bem mais sofisticado do que de seu irmão menor. Como o estado da transação deve ser preservado por um tempo determinado, esse componente precisa de mecanismos para salvar suas informações em um banco de dados.

- *Stateless Session Beans*: O componente de sessão distribuído não tem estado associado, logo permitem acessos concorrentes
- *Stateful Session Beans*: Os componentes de sessão distribuídos têm estado associado, no entanto, este estado não é persistente e o acesso a cada componente é limitado a um cliente.

Entity Beans

São objetos distribuídos com estado persistente. Este estado é imutável até mesmo pelo próprio componente.

Pode-se ter, como exemplo, um *Entity Bean* que represente a entidade aluno, implementando métodos para alterar e acessar os dados e mantendo a persistência das informações. Os *containers* cuidam das transações (*distribuídas ou não*) e existe um identificador único para cada entidade (*chave primária*).

Persistência é o atributo essencial das *entities bean*, podendo ser implementada pelo *bean* ou pelo *container*. Um *Entity Bean* implementa os métodos requeridos pelo *container* como *ejbCreate()*, *ejbFindByPrimaryKey*, *ejbLoad()*, *ejbStore()*.

Componentes em que o seu estado é controlado pelo seu *container* usam o *Container Managed Persistence - CMP*, ao passo que os componentes que mantêm o seu próprio estado usam *Bean Managed Persistence - BMP*, a tabela 2 mostra as diferenças entre o *BMP* e *CMP*.

Diferenças BMP-CMP

Diferença	Container-Managed Persistence	Bean-Managed Persistence
Definição da classe	Abstrata	Concreta
Chamadas de acesso ao banco de dados	Gerada pelas ferramentas no deployment	Codificada pelo programador
Estado persistente	Representadas como campos persistentes virtuais	Codificadas como variáveis de instância
Métodos de acesso a campos persistentes e relacionamentos	Obrigatórios (abstract)	Não há
Método findByPrimaryKey	Gerado pelo container	Codificado pelo programador
Métodos finder customizados	Gerados pelo container mas programador deve escrever EJB-QL	Codificado pelo programador
Métodos select	Gerados pelo container	Não há
Valor de retorno de ejbCreate()	Deve ser null	Deve ser a chave primária

Tabela 2: Diferenças BMP-CMP – (ROCHA, 2004)

2.4 – Serviço Distribuído e *Remote Method Invocation* - RMI

Para cada *EJB* instalado em um *container*, esse último automaticamente registra a *interface Home* do *EJB* em um serviço de diretório usando o *Java Name Directory Interface - JNDI* (SUN, 2004). Através do *JNDI*, os clientes então localizam o *EJB* que necessitam utilizar.

“ *JNDI* - A principal função de um serviço de nomes é permitir a associação de um nome (ou uma outra representação alternativa mais simples) a recursos computacionais como

- endereços de memória, de rede, de serviços
- objetos e referências
- códigos em geral

Suas duas funções básicas são

- Associar (mapear) um nome a um recurso
- Localizar um recurso a partir de seu nome.”

(ROCHA, 2004)

A tecnologia *EJB* usa o *Java Remote Method Invocation API (RMI)* para promover acessos a métodos remotos. O *RMI* suporta vários protocolos de comunicação (*IIOP, JRMP*).

Remote Method Invocation - RMI é uma das abordagens da tecnologia *Java* para prover as funcionalidades de uma plataforma de objetos distribuídos. Esse sistema de objetos

distribuídos faz parte do núcleo básico de *Java* desde a versão *JDK 1.1*, com sua *API* sendo especificada através do pacote *java.rmi* e seus subpacotes (*SUN, 2004*).

Através da utilização *RMI*, é possível que um objeto ativo em uma máquina virtual *Java* possa interagir com objetos de outras máquinas virtuais *Java*, independentemente da localização dessas máquinas virtuais.

No desenvolvimento de uma aplicação cliente-servidor usando *Java RMI*, como para qualquer plataforma de objetos distribuídos, é essencial que seja definida a interface de serviços que serão oferecidos pelo objeto servidor.

Os serviços especificados pela *interface RMI* deverão ser implementados através de uma classe *Java*. Nessa implementação dos serviços é preciso indicar que objetos dessa classe poderão ser acessados remotamente.

Com a *interface* estabelecida e o serviço implementado, é possível criar as aplicações cliente e servidor *RMI*.

A execução da aplicação cliente-servidor em *RMI* requer, além da execução da aplicação cliente e da execução da aplicação servidor, a execução do serviço de registro de *RMI*. Além do princípio básico de execução de aplicações *RMI*, a arquitetura *RMI* oferece facilidades para operação com código disponibilizado de forma distribuída e ativação dinâmica, além de outros serviços distribuídos.

RMI-IIOP – é uma versão da *RMI* implementada para usar o protocolo *CORBA IIOP*. *RMI* em cima de *IIOP* oferece interoperabilidade com objetos *CORBA* implementados em qualquer linguagem se as interfaces remotas foram definidas originalmente como interfaces *RMI* (*SUN, 2004*).

A *interface Home* do *EJB* define os métodos que permitem um cliente localizar e criar um *EJB Object*. Já a *interface Remote* estende *javax.ejb.EJBObject* e define os métodos que implementam a lógica de negócio que cliente pode chamar.

2.5 – Open Source

Pode-se definir o conceito fundamental de *open source* quando os programadores podem ler, redistribuir, e modificar o código fonte ou um pedaço de *software*. As pessoas contribuem e o aprimoram corrigindo *bugs* (entenda-se falhas). *Open* é um termo frequentemente mal entendido relativo a *software* grátis.

Programas que tem seu código aberto. Qualquer um pode baixar o código fonte do programa, estudá-lo ou mesmo aperfeiçoá-lo. *Open Source* não é a mesma coisa que de domínio público. Um programa *Open Source* continua pertencendo ao seu criador e a quem ajudou no seu desenvolvimento.

Open Source Initiative - OSI Web Site fornece recursos aos que definem os vários aspectos de *Open Source* inclusive uma definição de Fonte Aberta: <http://www.opensource.org/docs>. A referência seguinte da homepage OSI sintetiza os aspectos fundamentais:

"We in the open source community have learned that this rapid evolutionary process produces better software than the traditional closed model, in which only a very few programmers can see the source and everybody else must blindly use an opaque block of bits.

Open Source Initiative exists to make this case to the commercial world.

Open source software is an idea whose time has finally come. For twenty years it has been building momentum in the technical cultures that built the Internet and the World Wide Web. Now it's breaking out into the commercial world, and that's changing all the rules. Are you ready?" Fonte site <http://www.opensource.org/>

Segundo Bruce Perens, "*Open Source não significa apenas acesso ao código-fonte. Os termos de distribuição de softwares open source precisam seguir os seguintes critérios:*

1. Redistribuição livre

A licença não deve restringir qualquer grupo de vender ou oferecer o software como um componente de uma distribuição contendo programas de diversas fontes diferentes. A licença não deve cobrar royalties ou qualquer outro tipo de taxa por tal venda;

2. Código fonte

O programa deve incluir o código-fonte, e deve permitir a distribuição tanto no formato source code como no formato compilado. Quando alguma forma do produto é distribuída sem o código-fonte, deve haver uma maneira devidamente anunciada de como obtê-lo por não mais do que um custo razoável de reprodução (ex.: Custo de envio do CD com o código fonte) ou via download na Internet sem custos. O código-fonte deve ser a forma preferida pela qual um programador modificaria o programa. Código deliberadamente confuso não é permitido. Distribuição de formas intermediárias, tais como as saídas de um preprocessor não são permitidas;

3. Trabalho derivado

A licença deve permitir modificações e trabalhos derivados, e deve permitir que eles sejam distribuídos sob os mesmos termos da licença do software original;

4. Integridade do código-fonte do autor

A licença pode restringir o código-fonte de ser distribuído de maneira modificada somente se a licença permitir a distribuição de "patches" com a finalidade de modificar o programa em tempo de compilação. A licença deve explicitamente permitir a distribuição de software criado a partir do código fonte modificado. A

licença pode obrigar que trabalhos derivados tenham nome ou versão diferentes do software original;

5. Sem discriminação contra pessoas ou grupos

A licença não deve ser discriminatória contra nenhuma pessoa ou grupo de pessoas;

6. Sem discriminação contra campos de trabalho

A licença não deve restringir ninguém de fazer uso do programa em um campo específico de trabalho. Por exemplo, ela não pode restringir o programa de ser usado em uma determinada empresa, ou de ser usado em uma determinada pesquisa;

7. Distribuição da licença

Os direitos atribuídos ao programa se aplicam a todos para os quais o programa for redistribuído sem haver a necessidade da criação de uma licença adicional por essas partes;

8. A licença não deve ser específica de um produto

Os direitos atribuídos ao programa não devem depender do programa fazer parte de uma distribuição de software em particular. Se o programa for extraído da distribuição e usado ou distribuído dentro dos termos de sua licença, todos os grupos para quem o programa foi redistribuído devem ter os mesmos direitos que são garantidos em conjunto com a distribuição original;

9. A licença não deve restringir outros softwares

A licença não deve por restrições quanto ao uso de outro software distribuído com o software licenciado. Por exemplo, a licença não deve exigir que todos os outros programas distribuídos pelo mesmo meio devam ser Open Source" (PERENS, 1997).

2.5.1 – Library General Policy License - LGPL

Esta licença é derivada da *GPL*, que foi criada para atender a necessidade principalmente de bibliotecas desenvolvidas pela *Free Software Foundation*. Consiste em uma licença menos restritiva que a *GPL*, pois permite que o *software* desenvolvido sobre essa licença possa integrar um *software* comercial. Ela foi concebida para dar às bibliotecas *free* maior chance de competição com as *non-free*, e portanto melhores condições de se espalhar.

Na *GPL* "tradicional" todo o código do programa é aberto, isso atende bem à maioria dos projetos colaborativos. O problema é que muitas empresas possuem segredos a guardar, o que as impede de simplesmente abrir totalmente o código de seus programas.

Segundo Carlos A. M. dos Santos, professor da Universidade Regional Integrada em Santo Ângelo – RS, no que diz respeito à instalação, execução dos programas e aproveitamento dos resultados produzidos por eles, *BSD* e *GPL* se equivalem: regulam apenas

cópia, modificação e redistribuição do *software*. A *BSD* permite distribuição de código objeto ou executável, sem o código fonte.

A *GPL* exige que seja fornecido o código fonte, podendo-se cobrar pelo custo de reprodução, ou instruções de como obtê-lo (*dizer onde obter uma cópia via Internet, por exemplo*).

A *BSD* permite que o *software* seja incluído, no todo ou em parte, em outro *software* distribuído sob uma licença diferente. Com *GPL* é tudo ou nada: se alguém escrever um programa com milhares de linhas de código e incluir apenas algumas linhas de um código coberto pela *GPL*, o programa inteiro tem de ser distribuído sob *GPL*, a não ser que se obtenha permissão explícita para a cópia (*numa sutil contradição entre a licença e o preâmbulo, segundo o qual são as licenças de software comercial que nos privam da liberdade de compartilhar e modificar o software*).

Uma conseqüência desagradável disto é a falta de reciprocidade: pode-se incluir código distribuído sob licença *BSD* em *software* distribuído sob *GPL*, mas não o contrário. (*SANTOS, 2003*)

O código licenciado sob a *LGPL* (*pode-se usá-la, modificá-la e redistribuí-la livremente. É hoje uma das mais utilizadas para a produção de software livre, por ser muito completa, além de poder ser utilizada com várias linguagens de programação*) pode ser dinâmica ou estaticamente vinculado a qualquer outro código, independentemente da sua licença, assim como os usuários podem executar *debuggers* (*depurar um programa usualmente requer executar o programa e parar repetidamente em vários pontos durante a execução para examinar o valor de variáveis diferentes para determinar a causa de erros de lógica no programa*) no programa combinado. Na verdade, esta licença reconhece uma ligação entre o código *LGPL* e o código ao qual ele é vinculado.

2.6 – Extensible Markup Language - XML

É linguagem de marcação de dados (*meta-markup language*) que provê um formato para descrever dados estruturados. Isso facilita declarações mais precisas do conteúdo e resultados mais significativos de busca através de múltiplas plataformas. O *XML* também vai permitir o surgimento de uma nova geração de aplicações de manipulação e visualização de dados via *internet*.

O *XML* permite a definição de um número infinito de *tags*. Enquanto no *HTML* (W3C, 2004), se as *tags* podem ser usadas para definir a formatação de caracteres e parágrafos, o *XML* provê um sistema para criar *tags* para dados estruturados.

Um elemento *XML* pode ter dados declarados como sendo preços de venda, taxas de preço, um título de livro, a quantidade de chuva, ou qualquer outro tipo de elemento de dado. Como as *tags XML* são adotadas por *intranets* de organizações, e também via *Internet*, haverá uma correspondente habilidade em manipular e procurar por dados independentemente das aplicações onde os quais são encontrados. Uma vez que o dado foi encontrado, ele pode ser distribuído pela rede e apresentado em um *browser* como o *Internet Explorer* de várias formas possíveis, ou então esse dado pode ser transferido para outras aplicações para processamento futuro e visualização.

O *XML* provê uma representação estruturada dos dados que mostrou ser amplamente implementável e fácil de ser desenvolvida.

Implementações industriais na linguagem *SGML* (*Standard Generalized Markup Language*)(W3C, 2004) mostraram a qualidade intrínseca e a força industrial do formato estruturado em árvore dos documentos *XML*.

O *XML* é um subconjunto do *SGML*, o qual é otimizado para distribuição através da *web*, e é definido pelo *Word Wide Web Consortium* (W3C, 2004), assegurando que os dados estruturados serão uniformes e independentes de aplicações e fornecedores.

XML provê um padrão que pode codificar o conteúdo, as semânticas e as esquematizações para uma grande variedade de aplicações desde simples até as mais complexas, dentre elas:

- Um simples documento.
- Um registro estruturado tal como uma ordem de compra de produtos.
- Um objeto com métodos e dados como objetos *Java* ou controles *ActiveX*.
- Um registro de dados. Um exemplo seria o resultado de uma consulta a bancos de dados.
- Apresentação gráfica, como interface de aplicações de usuário.
- Entidades e tipos de esquema padrões.
- Todos os *links* entre informações e pessoas na *web*.

Uma característica importante é que uma vez tendo sido recebido o dado pelo cliente, tal dado pode ser manipulado, editado e visualizado sem a necessidade de reacionar o servidor. Dessa

forma, os servidores têm menor sobrecarga, reduzindo a necessidade de computação e reduzindo também a requisição de banda passante para as comunicações entre cliente e servidor.

O *XML* é considerado de grande importância na *Internet* e em grandes *intranets* porque provê a capacidade de interoperação dos computadores por ter um padrão flexível e aberto e independente de dispositivo. As aplicações podem ser construídas e atualizadas mais rapidamente e também permitem múltiplas formas de visualização dos dados estruturados.

Separação entre dados e apresentação: mais importante característica do *XML* se resume em separar a interface com o usuário (*apresentação*) dos dados estruturados. O *HTML* especifica como o documento deve ser apresentado na tela por um navegador. Já o *XML* define o conteúdo do documento. Por exemplo, em *HTML* são utilizadas *tags* para definir tamanho e cor de fonte, assim como formatação de parágrafo. No *XML* você utiliza as *tags* para descrever os dados, como exemplo *tags* de assunto, título, autor, conteúdo, referências, datas, etc... .

O *XML* ainda conta com recursos tais como folhas de estilo definidas com *Extensible Style Language (XSL)* e *Cascading Style Sheets(CSS)* para a apresentação de dados em um navegador. O *XML* separa os dados da apresentação e processo, o que permite visualizar e processar o dado como quiser, utilizando diferentes folhas de estilo e aplicações.

No *XML* as regras que definem um documento são ditadas por *DTDs (Document Type Definitions)*, as quais ajudam a validar os dados quando a aplicação que os recebe não possui internamente uma descrição do dado que está recebendo. Mas os *DTDs* são opcionais e os dados enviados com um *DTD* são conhecidos como dados *XML* válidos. Um analisador de documentos pode checar os dados que chegam analisando as regras contidas no *DTD* para ter certeza de que o dado foi estruturado corretamente. Os dados enviados sem *DTD* são conhecidos como dados bem formatados. Nesse caso, o documento pode ser usado para implicitamente se auto-descrever.

Com os dados *XML* válidos e com os bem-formatados, o documento *XML* se torna auto-descritivo porque as *tags* dão idéia de conteúdo e estão misturadas com os dados. Devido ao formato do documento ser aberto e flexível, ele pode ser usado em qualquer lugar onde a troca ou transferência de informação é necessária. Desta forma, podemos usar o *XML* para descrever informações sobre páginas *HTML*, ou descrever dados contidos em objetos ou regras de negócios, ou transações eletrônicas comerciais. O *XML* pode ser inserido dentro de documentos *HTML*, o que foi definido pelo *W3C* como "*data-islands*". Esse recurso permite

que um documento *HTML* possa ter múltiplas formas de visualização quando se faz uso da informação de semântica contida no *XML*.

O que define formalmente quais elementos e quais combinações possíveis são permitidas dentro de um documento *XML* é o "*schema*", ou seja, esquema. Existem novos esquemas propostos ao *W3C*, dentre eles estando o *DCD* (*Document Content Description*), que provêm à mesma funcionalidade dos *DTDs*, e que, pelo fato de linguagens esquema serem extensíveis, os desenvolvedores podem aumentá-los com informações adicionais, tais como regras de apresentação, tornando essas novas linguagens esquema mais poderosas que os *DTDs*.

As *DTDs* são formas de se descrever classes de documentos *XML* (como gramáticas para outras linguagens).

Problemas com *DTDs*:

- se muito simples não tem poder expressivo de descrição.
- se for muito complexa terá uma sintaxe horrível.

Um exemplo de *DTD*:

```
<!DOCTYPE recipecollection [
...
<!ELEMENT recipe
(title,author?,date?,description,ingredients,preparation,related)>
<!ATTLIST recipe id ID
#REQUIRED
category (breakfast|lunch|dinner|dessert|unknown)
#IMPLIED>
<!ELEMENT title (#PCDATA)>
<!ELEMENT author ANY>
...
]>
```

A solução para as *DTDs*: usar linguagens de esquemas (*schema languages*) tais como *DSD*, *XML Schema*, etc...

Os documentos, para serem validados, têm que ser bem formados e também estarem em conformidade com a *DTD* dada. (DOMINGUES, 2003)

2.7 – STUB – RPC

A comunicação entre processos em ambientes distribuídos pode ser feita basicamente através de memória compartilhada distribuída ou troca de mensagens. A primeira abordagem disponibiliza uma área de memória comum na qual os processos podem escrever e ler informações. A segunda técnica efetua transferência de dados entre processos através do

envio e recebimento de mensagens, diversas variantes do paradigma de troca de mensagens foram elaboradas como *Rendezvous (encontro)*, Chamada de Procedimento Remoto (*RPC*) e primitivas de *Send (envio)* e *Receive (resposta)*.

O modelo *Remote Procedure Call - RPC* é baseado na necessidade de se executar um componente de uma aplicação em qualquer local da rede. *RPCs* utilizam uma construção tradicional de programação - a chamada a procedimentos, a qual é estendida de um único sistema para uma rede de sistemas.

No contexto da comunicação em um ambiente cliente/servidor, a solicitação *RPC* de um determinado serviço de um componente de recurso (*servidor*) é emitida pelo componente de processos (*cliente*). O local do componente de recursos é transparente para o usuário (*cliente*). *RPCs* são muito utilizadas nas aplicações cliente/servidor, fornecendo ferramentas poderosas e necessárias ao desenvolvimento de programas distribuídos.

Stubs (adaptadores) são procedimentos que contém o código adicional ao programa para implementar *RPC*. Do lado do *software* que fará chamada (*cliente*), um procedimento *stub* substitui o procedimento que se tornou remoto. Do lado do procedimento que fará chamada (*servidor*), o *stub* substitui o mesmo que fará chamada. Estes dois *stubs* implementam toda a comunicação necessária para a chamada a procedimentos remotos, deixando os procedimentos originais intactos. (*IMASTER, 2001*)

Considerações finais deste capítulo

Neste capítulo foram abordados os conceitos relacionados ao *JBoss* introduzindo a tecnologia *J2EE*, sua arquitetura, *EJBs*, *JavaBeans*, *Container*, *Session Beans* e *Entity Beans*. Bem como uma introdução ao que vem a ser o conceito de *software Open Source* e a licença *LGPL*, uma breve definição de *XML* e para finalizar conceitos de *STUB-RPC*.

3 – JBOSS APPLICATION SERVER

3.1- Introdução

O desenvolvimento do *JBoss* iniciou-se em março de 1999. Nascido como um simples *container EJB* e, ao longo dos anos, evoluiu para ser um servidor de aplicações *Java* completo. Ele é desenvolvido por uma comunidade *open source* sob a licença *LGPL* e está se tornando um sério concorrente aos servidores de aplicação comercial.

O Grupo *JBoss* (*JBOSS, 2004*) recentemente lançou a versão 4.x do seu servidor de aplicações *Java* que suporta a especificação *Java 2 Enterprise Edition — J2EE* da *Sun Microsystems*.

O *JBoss* conquistou em 2002 o prêmio da *JavaWorld* na categoria “*Best Java Application Server*”, estando à frente de produtos comerciais cujas licenças custam vários milhares de dólares por *CPU*. (*JBOSS, 2004*)

Uma funcionalidade bastante importante é que o *microkernel* baseado em *JMX* (ver capítulo 3, seção 3.7) pode, ao ser inicializado, baixar toda a sua configuração, as classes de que necessita e as aplicações a partir de um servidor *HTTP*. Com isto é possível se criar *farms* (redes de servidores) de servidores *JBoss* a partir de um servidor central. E como o *microkernel* do *JBoss* cabe tranqüilamente em um *floppy (disquete)*, é possível fazer o servidor totalmente autoconfigurado.

3.2 – JBoss Clustering

Clustering é um serviço de nós, estes nós geralmente têm finalidades em comum:

- ü Tolerância à Falhas;
- ü Balanceamento de Carga por réplica.

Estes conceitos estão freqüentemente misturados. Um nó pode ser um computador ou, mais simplesmente, um exemplo de servidor (*se for servidor de diversas ocorrências*). (*LABOUREY; BURKE, 2002*)

3.3 – Terminologia de *Clustering* em *JBoss*

Disponibilidade de um serviço é uma proporção de tempo para o qual um serviço é acessível, com tempos de resposta razoável/previsível particular. O termo Alta Disponibilidade geralmente é usado para denotar uma “*proporção*” alta. Não obstante, esta proporção é contexto-dependente: Alta Disponibilidade - *HA* para um sistema crítico em um espaço provavelmente, está baseado na figura mais alta de *HA* para um *site* da *web* regional. A *HA* proporciona assim permissão máxima de tempo para manutenção em um período particular.

A tabela 3 apresenta alguns exemplos de permissão máxima para tempos de manutenção por ano que depende da proporção de *HA*.

HA Proporção	Permissão máxima de tempo para manutenção acumulado por ano
98 %	7.3 dias
99 %	87.6 horas
99.5 %	43.8 horas
99.9 %	8.76 horas
99.95 %	4.38 horas
99.99 %	53 minutos
99.999 %	5.25 minutos
99.9999 %	31 segundos
99.99999 %	3.1 segundos

Tabela 3 - Amostra permissão para tempo de manutenção por proporções de *HA*

Está claro que até mesmo se a proporção de *HA* for estritamente relativa ao seu tempo de manutenção permitido associado, custo geralmente não for: passando de 99% a 99.99% é geralmente muito mais caro do que 98% a 99% até mesmo se a diferença for maior.

Por exemplo, as Empresas de telecomunicações geralmente requerem uns "5-9" (e.x. 99.999%) nível de *HA*.

- Tolerância à falhas implica Alta Disponibilidade. Não obstante, dados Altamente Disponíveis não são dados necessariamente corretos, considerando que uma falta de serviço tolerante sempre garante comportamento estritamente correto apesar de um certo número e tipo de falhas.

Conseqüentemente, alguns sistemas só requerem alta disponibilidade (*serviço de diretório que consiste por exemplo em dados estáticos*) considerando que outros requerem tolerância à falhas (*sistemas bancários que requerem confiança transacional por exemplo.*)

- Balanceamento de carga. (ver capítulo 3, seção 3.6).
(LABOUREY; BURKE, 2002)

Alta disponibilidade pode ser definida como redundância. Se um servidor falhar ou não puder atender uma requisição, então outro servidor assumirá, da forma mais transparente possível, o processamento da requisição. Isso tende a eliminar os pontos de falha de uma aplicação.

3.4 – Características de *Clustering JBoss*

JBoss atualmente suporta as seguintes características de *clustering*.

- *Automatic Discovery*. Nós à procura de *clustering* ao outro sem configuração adicional.
- *Fail-Over* e *Load-balancing* - características para:
 - *JNDI*,
 - *RMI* (pode ser usado para implementar seus próprios serviços agrupados),
 - *Entity Beans*,
 - *Stateful Session Beans* com estado de memória replicante,
 - *Stateless Session Beans*
- *HTTP Session replica* com *Tomcat (3.0)* e *Jetty (CVS HEAD)*
- *Dynamic JNDI discovery*. Clientes de *JNDI* podem descobrir o *JNDI InitialContext* automaticamente.
- *Cluster-wide replicated JNDI tree*
- *Farming - Cluster-wide hot-deployment distribution*
- *Pluggable RMI load-balance policies*.

3.5 – Partições

Partição é o conceito central no que diz respeito a *clustering* em *JBoss*.

Em uma mesma rede, pode-se ter partições diferentes. Para distingui-las, cada partição tem que conter um nome particular.

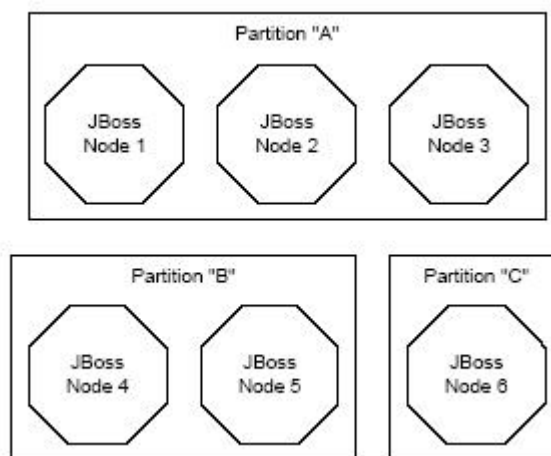


Figura 4 - Partições. (LABOUREY; BURKE, 2002)

Na figura 3, uma das amostras de *clustering* um caso de limite: está composto de um único nó. Enquanto isto não traz interesse particular (*nenhuma tolerância à falhas ou balanceamento de carga é possível*), um nó novo poderia ser acrescentado qualquer hora a esta partição que ficaria muito mais interessante então.

É possível uma instancia do *JBoss* fazer ao mesmo tempo parte de partições múltiplas; consideraremos que um servidor *JBoss* sempre faz no entanto parte de uma única partição. Se nenhum nome é atribuído a uma partição, então, é utilizado um nome padrão.

3.6 – Balanceamento de Carga

É uma média para obter melhor performance, despachando pedidos que chegam a servidores diferentes. Não faz nenhuma suposição no nível de tolerância à falhas ou disponibilidade do sistema

O crescimento constante da *Internet* vem causando diversos problemas de desempenho, incluindo baixos tempos de resposta, congestionamento da rede e interrupção de serviços (*DOS*). Existem diversas abordagens de como esses problemas podem ser contornados. Exemplo de alguns deles são:

- Espelhamento do *Site* - em diversos locais que podem ser acessados manualmente pelos usuários através de uma listagem com as *URLS* correspondentes. Esse tipo de solução traz diversas desvantagens

como a não transparência ao usuário e ausência de controle na distribuição de requisições.

- Servidores *Proxy* - consiste em manter cópias (*caches*) de objetos *Web* acessados perto dos usuários. Isso pode ser controlado por servidores que colocam objetos *Web* populares em outros servidores cooperativos ou ainda disparados por requisições individuais de usuários que passam por um servidor *Proxy*. Uma outra técnica consiste em pré-carregar os objetos frequentemente acessados de forma a mascarar a latência da rede.
- Balanceamento de Carga - é fazer o servidor *Web* mais poderoso através do uso de uma arquitetura em *cluster* na qual múltiplas máquinas funcionam como um único servidor. Um *cluster* é definido como um grupo de servidores executando a mesma aplicação *Web* simultaneamente, aparecendo para o mundo como se fosse um único servidor.

Para balancear a carga nos servidores, o sistema distribui as requisições para diferentes nós que compõem o *cluster* de servidores, com objetivo de otimizar o desempenho do sistema. Os resultados são:

- Alta disponibilidade - (*ver capítulo 3, seção 3.3*);
- Escalabilidade - é a habilidade que uma aplicação tenha de suportar um crescente número de usuários, ou seja no contexto, é uma medida de vários fatores, incluindo o número de usuários simultâneos que um *cluster* pode suportar e o tempo que se leva para responder uma requisição;
- Administração facilitada da aplicação - no sentido de que o *cluster* aparece como um único sistema para os usuários, aplicações e para o resto da rede, facilitando o acesso e administração do sistema e dos recursos de rede.

Com respeito às entidades que podem realizar o balanceamento de carga, temos:

- Baseada no Cliente;
- Baseada no *DNS*;

- Baseada num Despachante;
- Baseada no Servidor.

Existem duas abordagens de como colocar o mecanismo de seleção de servidores no lado do cliente satisfazendo o requisito de transparência: através dos próprios clientes (*browsers*) ou por meio de servidores *Proxy*

Tipicamente a transparência da arquitetura é obtida através de uma única interface virtual direcionada ao mundo externo, pelo menos ao nível da *URL*.

Numa primeira solução desenvolvida no lado do *cluster*, a responsabilidade de distribuir as requisições entre os servidores é atribuída ao *DNS* do *cluster*, ou melhor, ao servidor de *DNS* autoritário pelo domínio dos nós do *cluster*. Através de um processo de tradução entre os nomes simbólicos (*URL*) e endereços *IP*, o *DNS* do *cluster* pode selecionar qualquer nó que compõe o *cluster*.

Uma abordagem alternativa à arquitetura baseada em *DNS* visa ter total controle sobre as requisições de clientes e mascarar o roteamento entre múltiplos servidores. Para esse propósito, a virtualização do endereço, realizada na solução baseada em *DNS* é estendida do nível da *URL* para o nível do *IP*. Nessa abordagem um único endereço *IP* virtual é fornecido ao *cluster Web*. Esse é o endereço do chamado "*despachante*" que atua com escalonador central do *cluster*.

3.7 – *Java Management Extension - JMX*

Conhecida anteriormente por *JMAPI*, define uma arquitetura de gerência, *APIs*, e serviços de gerência, todos sobre uma única especificação. A especificação *JMX* foi desenvolvida pela *SUN* em parceria com os principais líderes da indústria de gerência, seguindo a Comunidade *Java* (*SUN, 2004*).

JMX fornece uma maneira simples para instrumentação de objetos *Java*. A instrumentação *JMX* tem poucas limitações porque é totalmente independente da infraestrutura de gerência. Isto significa que um recurso pode ser gerenciado sem a preocupação de como seu gerente é implementado, se é implementado, por exemplo, sobre *TMN* ou *SNMP*. (*SUN, 2004*)

JMX permite que os desenvolvedores de aplicações baseadas em tecnologia *Java* criem os agentes inteligentes e gerentes na linguagem *Java*. Estas aplicações podem ser

integradas às soluções em sistemas de gerência existentes. A arquitetura *JMX* é dividida em três níveis: nível de instrumentação, nível de agente e nível de gerente.

A especificação do *JMX* é um padrão para gerenciar redes, aplicações, dispositivos, etc., através de *Java*. O *JMX* é uma extensão aberta e universal da linguagem *Java* e permite que corporações e provedores de serviços gerenciem ambientes heterogêneos de uma maneira padrão. (MIC99, 1999).

3.8 Arquitetura *JMX*

A divisão dos níveis traz flexibilidade, permitindo que subconjuntos da especificação sejam utilizados individualmente por diferentes comunidades de desenvolvedores que utilizam a t

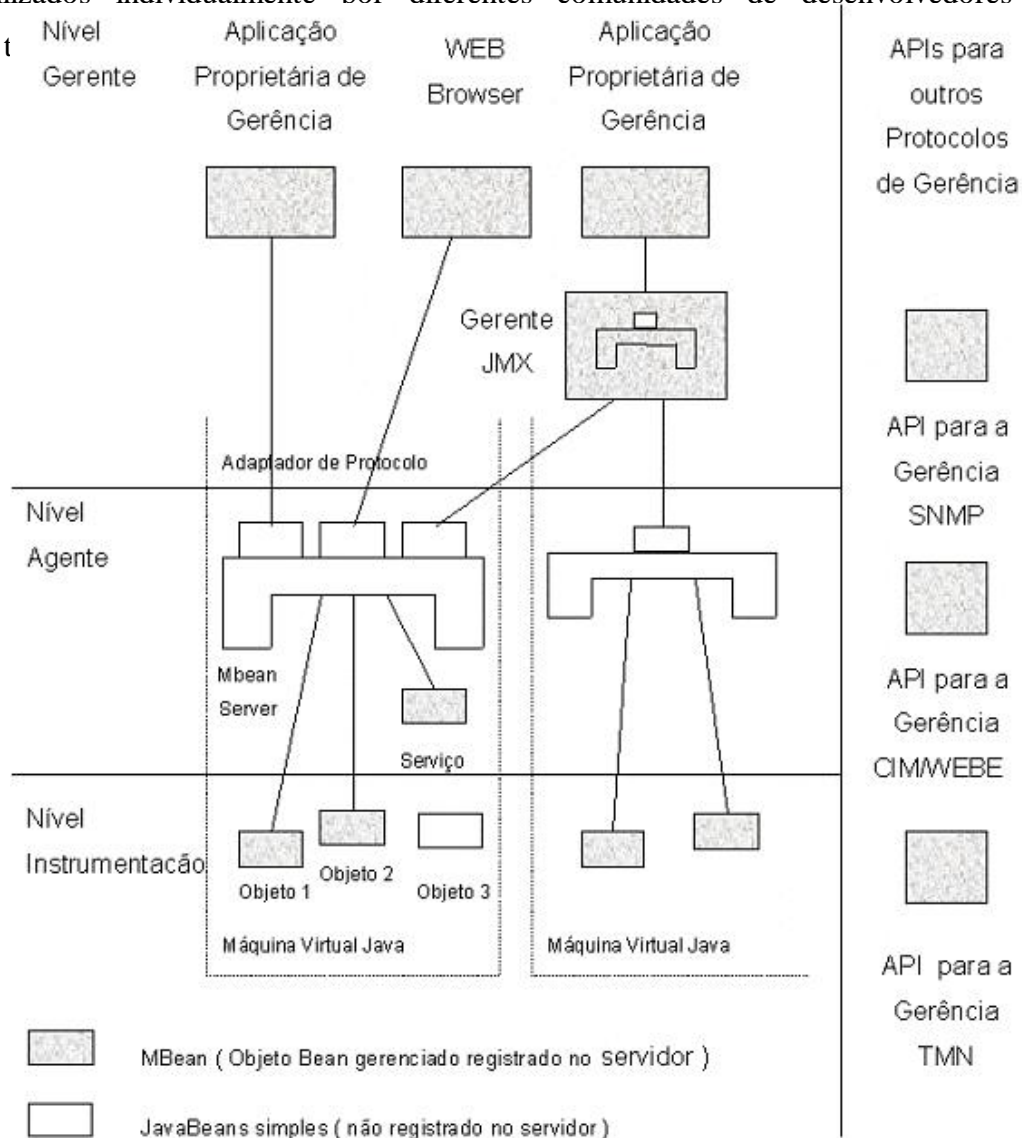


Figura 5 - Arquitetura JMX

O nível de instrumentação fornece a gerência imediata de qualquer objeto baseado em tecnologia *Java*. Este nível é direcionado a toda a comunidade de desenvolvedores que utiliza tecnologia *Java*.

O nível agente fornece os agentes de gerência. Os agentes *JMX* são recipientes que contêm a base dos serviços de gerência. Esta base pode ser facilmente estendida adicionando-se recursos *JMX*. Este nível é direcionado para a comunidade de desenvolvedores de soluções de gerência e fornece o gerenciamento através da tecnologia *Java*.

O nível gerente fornece os componentes de gerência que podem operar como gerente ou agente para distribuição e consolidação dos serviços de gerência. Este nível é direcionado para a comunidade de desenvolvedores de soluções de gerência e a complementa através da tecnologia *Java* provida pelo nível agente. As *APIs* de protocolos de gerência adicionais são direcionadas para a comunidade de desenvolvedores de soluções de gerência e proporcionam a integração com as soluções já existentes.

O *JMX* possui ainda componentes como: (*SUN, 2004*)

- Recurso Gerenciável
- Agente *JMX*
- Gerente *JMX*
- Serviços de Gerência
- *APIs* para outros protocolos de gerência

3.9 – Segurança em *JBOSS*

Segurança é uma parte fundamental de qualquer aplicação de uma empresa. Pode-se restringir o que é permitido acessar em suas aplicações, e controle de usuários em que aplicações de operações podem executar. As especificações de *J2EE* definem um modelo de segurança *role-based* simples para *EJBs* e componentes *Web*. O componente *JBoss framework* que controla segurança é o *JBossSX extension framework* (*JBOSSSX, 2004*).

Um *framework* é uma aplicação reusável, semicompleta que pode ser especializada para produzir aplicações customizadas. Por exemplo, para criar a *interface* de um aplicativo é necessário instanciar diversos objetos, tais como: formulários, botões, caixas de textos, menus, etc. Cada ocorrência de cada um desses elementos na *interface* do aplicativo é um

objeto. O módulo de *software* que contém as classes prefabricadas a partir das quais se criam esses objetos é o *framework* de classes (*class framework*).

Diz-se que o *framework* é uma aplicação porque contém a implementação de diversas classes de objetos que estão relacionadas entre si e pertencem a um mesmo assunto, como por exemplo interface gráfica; diz-se que ele é reusável porque pode ser utilizado na construção de diferentes sistemas; semi-completo porque nunca implementa tudo o que seria possível; e customizável porque admite que o desenvolvedor estenda suas funcionalidades.

Existem *frameworks* em diversas áreas, tais como: acesso a banco de dados, persistência de objetos em ambiente relacional, criação de *interface* gráfica, gerenciamento de coleções de objetos, segurança, etc.

3.9.1 – *J2EE Declarative Security Overview*

O modelo de segurança defendido pela especificação *J2EE* é um modelo declarativo, pois se descreve os papéis de segurança e permissões que usam um descriptor *XML standard* em lugar de embutir segurança em seu componente *business*. Isto isola a segurança do código de *business-level* porque a segurança tende a ser mais uma função onde o componente é desdobrado, em lugar de um aspecto inerente da lógica de negócio do componente. Por exemplo, considere um componente de BANCO 24 HORAS que será usado para acessar uma conta bancária. As exigências de segurança, papéis e permissões variarão independente de como a pessoa acessa a conta bancária baseada em que banco está administrando a conta onde o banco 24 horas é desdobrado, e assim por diante.

3.9.2 – *Secure Remote Password (SRP) Protocol*

O protocolo de *SRP* é uma implementação do *public key exchange handshake* descrita nos padrões de *Internet (RFC2945)*.

O *framework* de *JBossSX* inclui uma implementação de *SRP* que consiste nos elementos seguintes:

- Implementação do *SRP handshake protocol* que é independente de qualquer protocolo *client/server* particular.

- Implementação *RMI* do protocolo handshake como o *client/server* por padrão *SRP* implementado.
- Ao lado do cliente implementação *JAAS LoginModule* que usa a implementação de *RMI*, para uso autenticando os clientes em um modo seguro .
- *JMX MBean* por administrar o servidor de implementação *RMI*. O *MBean* permite ao servidor de implementação *RMI* conectar-se em um *framework JMX* e externaliza a configuração de armazenamento, informação e verificação. Também estabelece um *cache* de autenticação que é ligado no servidor *JBoss JNDI namespace*.
- Ao lado do servidor *JAAS LoginModule implementation* que usa o *cache* de autenticação administrada pelo *SRP JMX MBean*. (*JBOSSSX, 2004*)

Considerações finais deste capítulo

Neste capítulo foram abordados os seguintes contextos: quando se deu início ao *JBoss*, sua versão atual e alguns destaques obtidos. Foram citadas as metas comuns do *clustering* com balanceamento de carga e tolerância a falhas características e sobre partições do *Clustering JBoss*, quais os resultados que podem ser obtidos utilizando-se deste recurso, algumas alternativas de como melhorar o desempenho utilizando *cluster*.

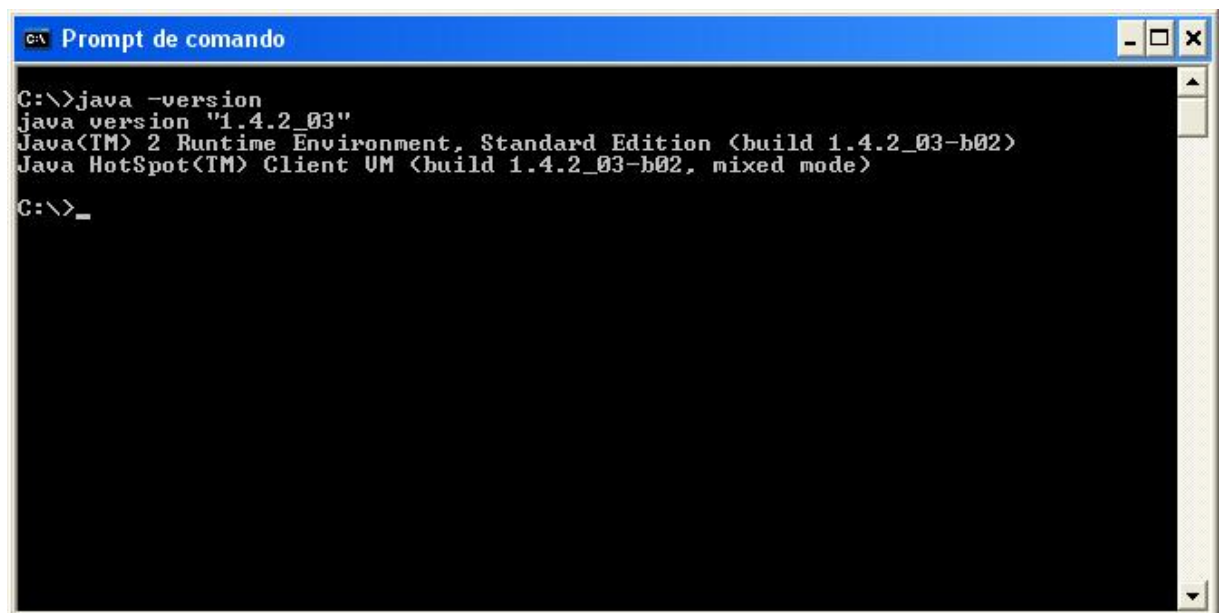
Conceitos básicos do *Java Management Extension – JMX*, sua arquitetura e componentes e a utilização de um modelo de três níveis, instrumentação, agente e gerente *JMX* e finalizando o capítulo alguns conceitos de segurança no *JBoss*.

4 – INSTALANDO E CONSTRUINDO O SERVIDOR *JBoss*

A seguir será abordada a instalação básica do *JBoss*, requisitos para instalação, dicas e como executar o servidor.

4.1 – Condições prévias para instalação

Antes da instalação e execução do Servidor *JBoss* é necessário *JSDK 1.3* ou superior (SUN, 2004). Pode-se fazer o *download* em <http://java.sun.com/j2se/1.4.2/download.html>. Atualmente, a versão mais atual é a *1.4.2_02*, na seção *Download J2SE v 1.4.2_02* e faça o *download* de *Windows Installation (SDK)*. Para verificar este requisito, é necessário executar o comando “*java-version*” para assegurar que o *Java Executable*, encontra-se no *path* conforme a figura 4.



```
C:\>java -version
java version "1.4.2_03"
Java(TM) 2 Runtime Environment, Standard Edition (build 1.4.2_03-b02)
Java HotSpot(TM) Client VM (build 1.4.2_03-b02, mixed mode)
C:\>_
```

Figura 6. Execução do comando *java -version*

JBoss possui um banco de dados relacional nativo escrito em *Java*, o *Hypersonic SQL* (*POINTBASE, 2004*). Isso facilita a vida do desenvolvedor na hora de testar seus *Entity Beans* eliminando assim o trabalho de configurar uma nova base de dados. Para instalar uma nova base de dados no *JBoss* é necessário seguir alguns passos. A configuração e instalação da base de dados são feitas através de um arquivo *XML Metadata Interchange*.

No diretório *\$JBoss_DIST/docs/examples/jca* conforme a figura 7 há vários modelos de arquivos de configuração.

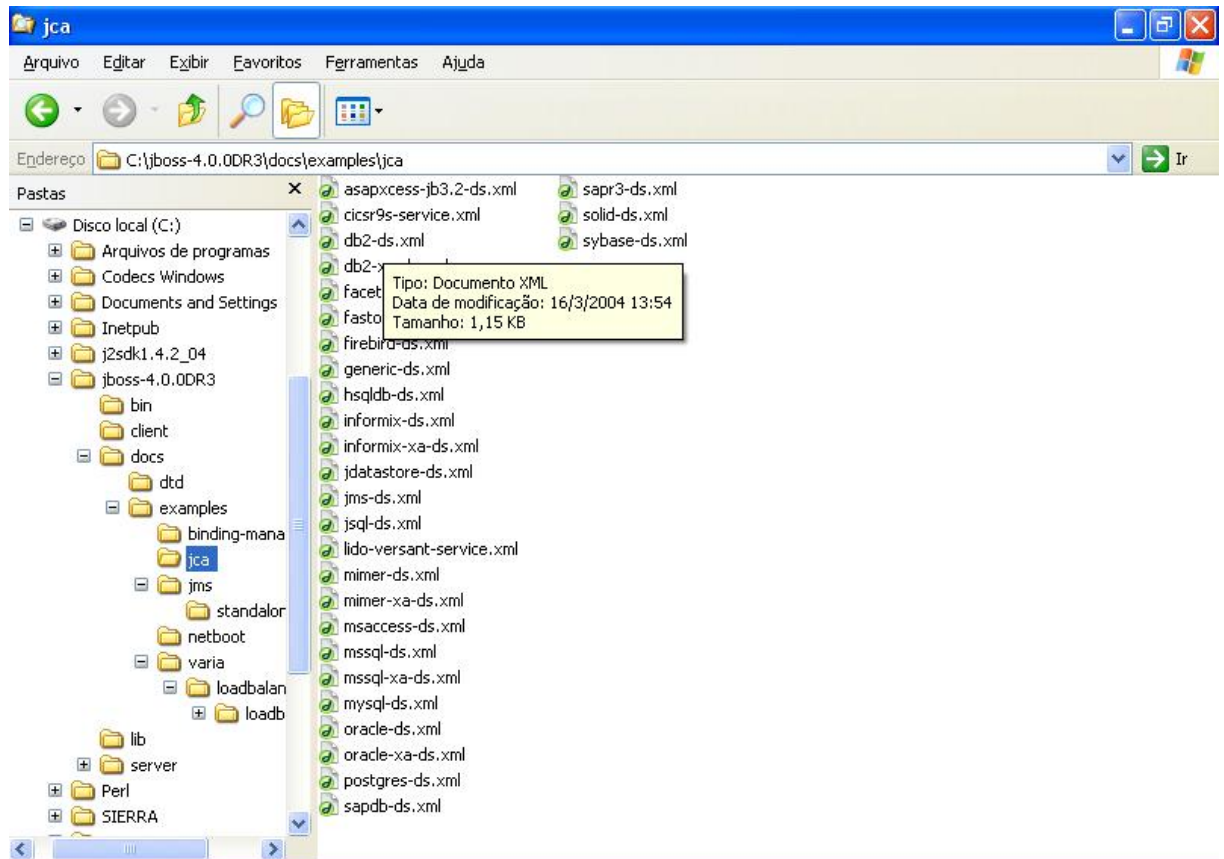


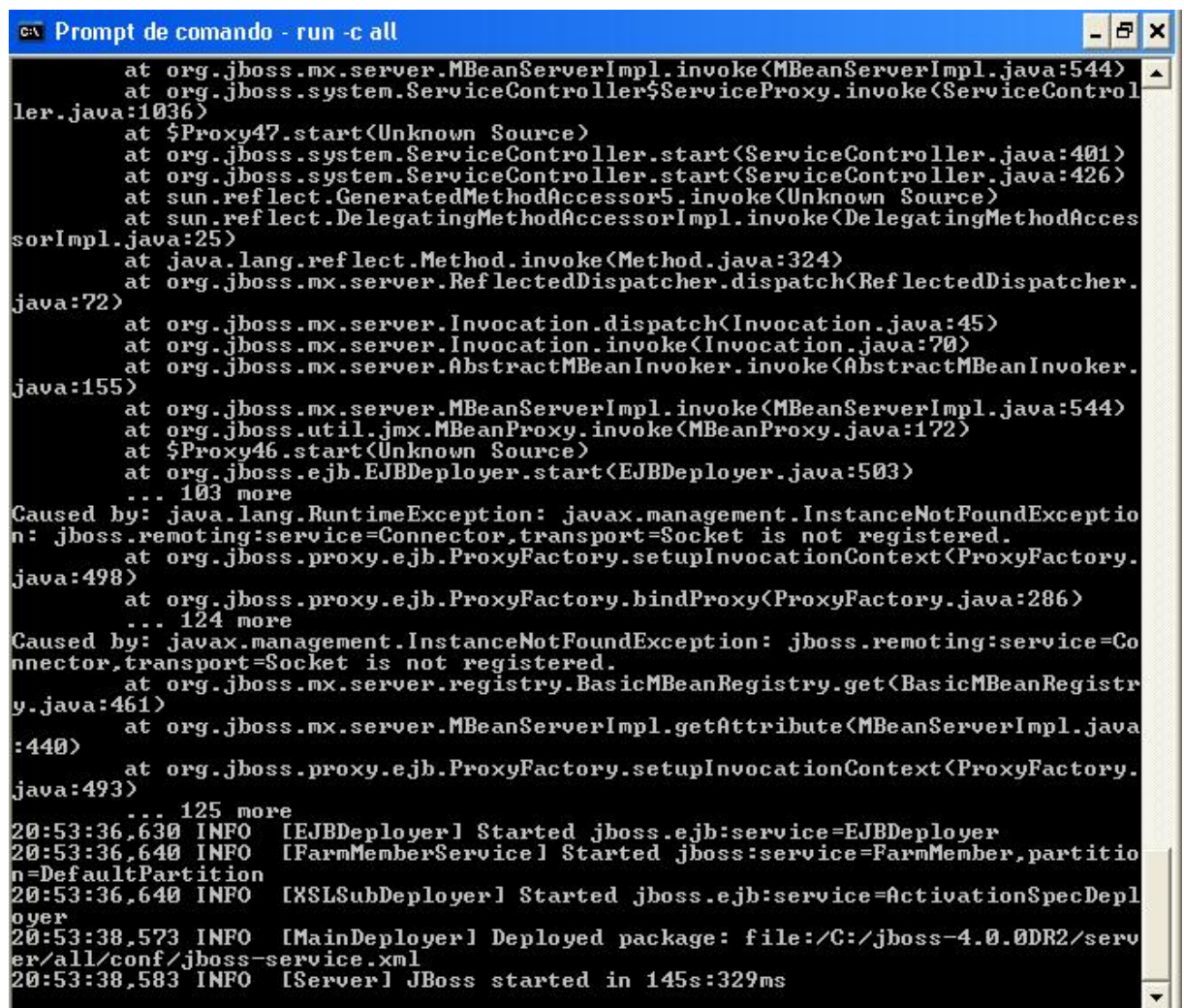
Figura 8. Modelos de arquivos para configuração

Uma sugestão para *WebContainer* no *JBoss* pode-se optar entre *Tomcat* (*JAKARTA, 2004*) - é mais que um servidor de aplicações com as características de servir como um controlador de *servlets* e *JSP* e com a vantagem de ser gratuito)) ou *Jetty* (*JETTY, 2003*) - também um *MBean*). A grande sugestão fica por conta do *Jetty* que há uma ótima integração com o *JBoss*, ambos podem ser executados na mesma *Java Virtual Machine* (*JVM* permite que programas desenvolvidos em *Java* sejam executados em qualquer computador, independente do sistema operacional ou do hardware. O sistema é ofertado por diversos fornecedores, incluindo a *Sun*).

4.2 – Configurando e Iniciando o *JBoss*

Após o *download*, descompacte o arquivo na raiz da unidade, ex: *C:\JBoss-4.0.0DR3.*, pode-se usar umas das três configurações pré-estabelecidas pelo *JBoss*, ou o mais recomendado é criar uma própria configuração.

Para iniciar o *JBoss*, é necessário acessar o diretório *\$JBASS_DIST/bin* e executar o arquivo de lote *run*. Para os usuários do *Windows* a linha de comando é *%JBASS_DIST%\bin\run.bat* e para os usuários do *Linux* a linha é *\$JBASS_DIST/bin/run.sh*.



```

at org.jboss.mx.server.MBeanServerImpl.invoke(MBeanServerImpl.java:544)
at org.jboss.system.ServiceController$ServiceProxy.invoke(ServiceControl
ler.java:1036)
at $Proxy47.start(Unknown Source)
at org.jboss.system.ServiceController.start(ServiceController.java:401)
at org.jboss.system.ServiceController.start(ServiceController.java:426)
at sun.reflect.GeneratedMethodAccessor5.invoke(Unknown Source)
at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAcces
sorImpl.java:25)
at java.lang.reflect.Method.invoke(Method.java:324)
at org.jboss.mx.server.ReflectedDispatcher.dispatch(ReflectedDispatcher.
java:72)
at org.jboss.mx.server.Invocation.dispatch(Invocation.java:45)
at org.jboss.mx.server.Invocation.invoke(Invocation.java:70)
at org.jboss.mx.server.AbstractMBeanInvoker.invoke(AbstractMBeanInvoker.
java:155)
at org.jboss.mx.server.MBeanServerImpl.invoke(MBeanServerImpl.java:544)
at org.jboss.util.jmx.MBeanProxy.invoke(MBeanProxy.java:172)
at $Proxy46.start(Unknown Source)
at org.jboss.ejb.EJBDeployer.start(EJBDeployer.java:503)
... 103 more
Caused by: java.lang.RuntimeException: javax.management.InstanceNotFoun
dException: jboss.remoting:service=Connector,transport=Socket is not registered.
at org.jboss.proxy.ejb.ProxyFactory.setupInvocationContext(ProxyFactory.
java:498)
at org.jboss.proxy.ejb.ProxyFactory.bindProxy(ProxyFactory.java:286)
... 124 more
Caused by: javax.management.InstanceNotFoundException: jboss.remoting:service=Co
nconnector,transport=Socket is not registered.
at org.jboss.mx.server.registry.BasicMBeanRegistry.get(BasicMBeanRegistr
y.java:461)
at org.jboss.mx.server.MBeanServerImpl.getAttribute(MBeanServerImpl.java
:440)
at org.jboss.proxy.ejb.ProxyFactory.setupInvocationContext(ProxyFactory.
java:493)
... 125 more
20:53:36,630 INFO [EJBDeployer] Started jboss.ejb:service=EJBDeployer
20:53:36,640 INFO [FarmMemberService] Started jboss:service=FarmMember,partitio
n=DefaultPartition
20:53:36,640 INFO [XSLSubDeployer] Started jboss.ejb:service=ActivationSpecDepl
oyer
20:53:38,573 INFO [MainDeployer] Deployed package: file:/C:/jboss-4.0.0DR2/ser
ver/all/conf/jboss-service.xml
20:53:38,583 INFO [Server] JBoss started in 145s:329ms

```

Figura 9. Tempo para carregamento do servidor *JBoss*

O teste acima demonstrado na figura 8 o tempo de carregamento do servidor *JBoss* com a configuração completa (*all*), isto sendo a primeira execução, foi de 145s:329ms em um equipamento com a seguinte configuração *Pentium III 800 MHz, 512 MB Ram, HD 40 GB* com sistema operacional *Windows XP PRO*.

Pode-se nomear a configuração do servidor escolhendo uma configuração padrão ou criando uma personalizada. O *JBoss* vem com três configurações padrão: mínima (*minimal*), padrão (*default*) e completa (*all*). Estas configurações estão em *\$JBOSS_DIST/server*. O usuário pode criar ainda a sua própria configuração, bastando para isso apenas criar um novo diretório junto com as outras configurações. É sempre melhor criar a sua própria configuração, pois isso facilita manutenção.

Para visualizar o estado dos componentes do *JBoss (MBeans)*, deve-se abrir o *browser* no endereço <http://localhost:8082>. O endereço padrão do *WebServer (Jetty ou Tomcat)* é <http://localhost:8080> conforme figura 10. Na versão 3.0.2 em diante o endereço para visualizar os componentes *JMX* é <http://localhost:8080/jmx-console> conforme a figura 9.

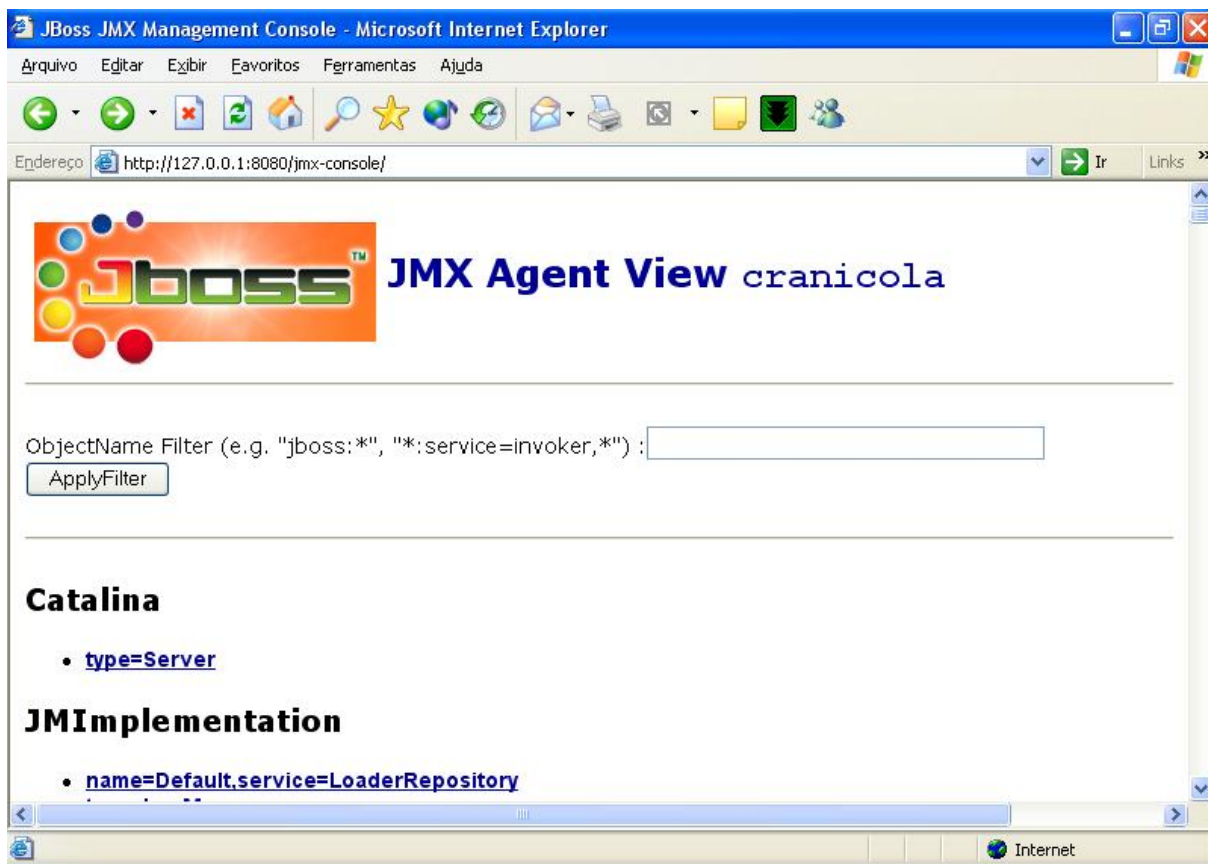


Figura 10. Mostra tela de configuração *jmx-console* do servidor *JBoss*

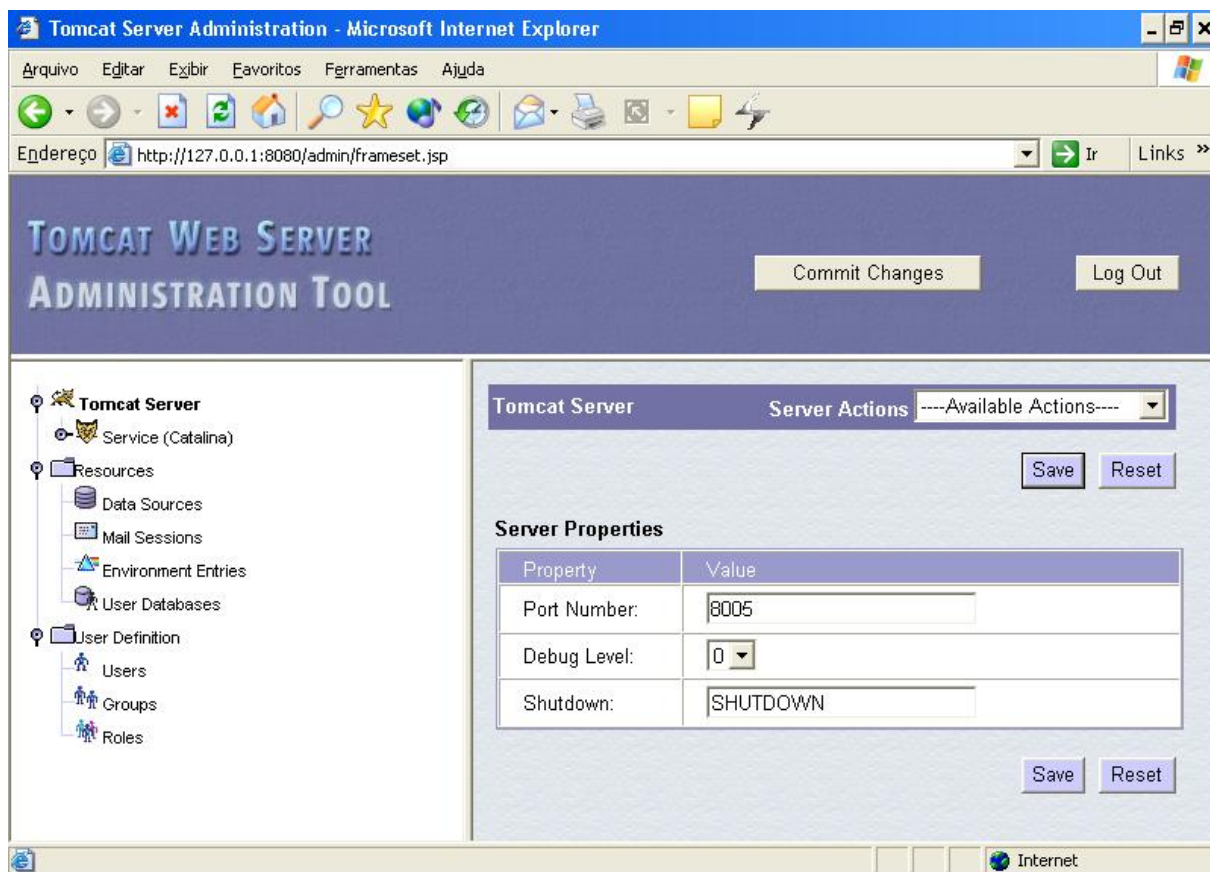


Figura 11. Mostra tela de configuração do *TomCat*

4.3 – Estrutura de diretórios

A distribuição do *JBoss* cria um diretório *jboss-4.x.x* que contém *server start scripts*, *jars* (Java Archive - um formato de arquivo independente de plataforma, permitindo que muitos arquivos sejam agregados em um único arquivo), configuração de servidor e diretórios de trabalho.

Uma instalação padrão do *JBoss* inclui os seguintes diretórios imediatamente abaixo do diretório superior *< jboss-home >*, como mostrado na figura 6.

- *BIN*: Contém todos os arquivos executáveis (*both scripts* e *JARs*) incluídas na distribuição *JBoss*.

- *CLIENT*: Este é o diretório onde as bibliotecas requeridas para clientes são colocadas. Um cliente típico requer *jboss-client.jar*, *jbossx-client.jar*, *jaas.jar*, *jnp-*

client.jar, *ejb.jar* e *jta-spec1_0_1.jar*. Se o cliente não estiver rodando *JDK 1.3*, requererá *jndi.jar* também.

- *DOCS*: Contêm a documentação *JBoss API*, o *Javadoc-style*, e a outra documentação no formato do *HTML*.

- *LIB*: Contêm bibliotecas *java* no formato *JAR* que o *JBoss* usa. O diretório *lib*, contém arquivos *JARs* que necessitam estar no *path* da classe do sistema; os *JARs* em *lib* estão disponíveis ao classloader *Mlet-baseado* servidor *JBoss*.

- *SERVER*: cada um dos subdiretórios dentro é aqui uma configuração diferente do usuário. A configuração é selecionada passando "*-c < nome da opção da configuração >*" ao *script*.

- *LOG*: Os *logs* de registro do *JBoss* estão situados neste diretório. Registrar os *logs* é iniciado por padrão.

- *DEPLOY*: Este é o diretório da distribuição. Onde são colocados os arquivos *JAR* e *EAR* e aqui serão desdobrados automaticamente.

- *CONF*: O conjunto de configuração *JBoss* é encontrado aqui. Por padrão há somente um conjunto de configuração, situado no subdiretório padrão, adicionar mais de é permitido. A instalação empacotada do *JBoss* com *web container* (*Tomcat* ou *Jetty*) cria um conjunto adicional de configuração.

- *DB*: O diretório que contém outros diretórios com arquivos relacionados às bases de dados *Hypersonic* e *Instantdb* (*configuração, indexação, tabelas, etc..*)

- *DEPLOY*: desdobra o código de aplicação (*jar, war e ear files*) e os baixa aqui. Também é usado para serviços *hot-deployable* (*esses aos quais podem ser acrescentados ou removidos do servidor corrente*) e desdobrar *JCA resource adapters3*.

A figura 11 mostra a estrutura de diretórios do servidor *JBoss*.

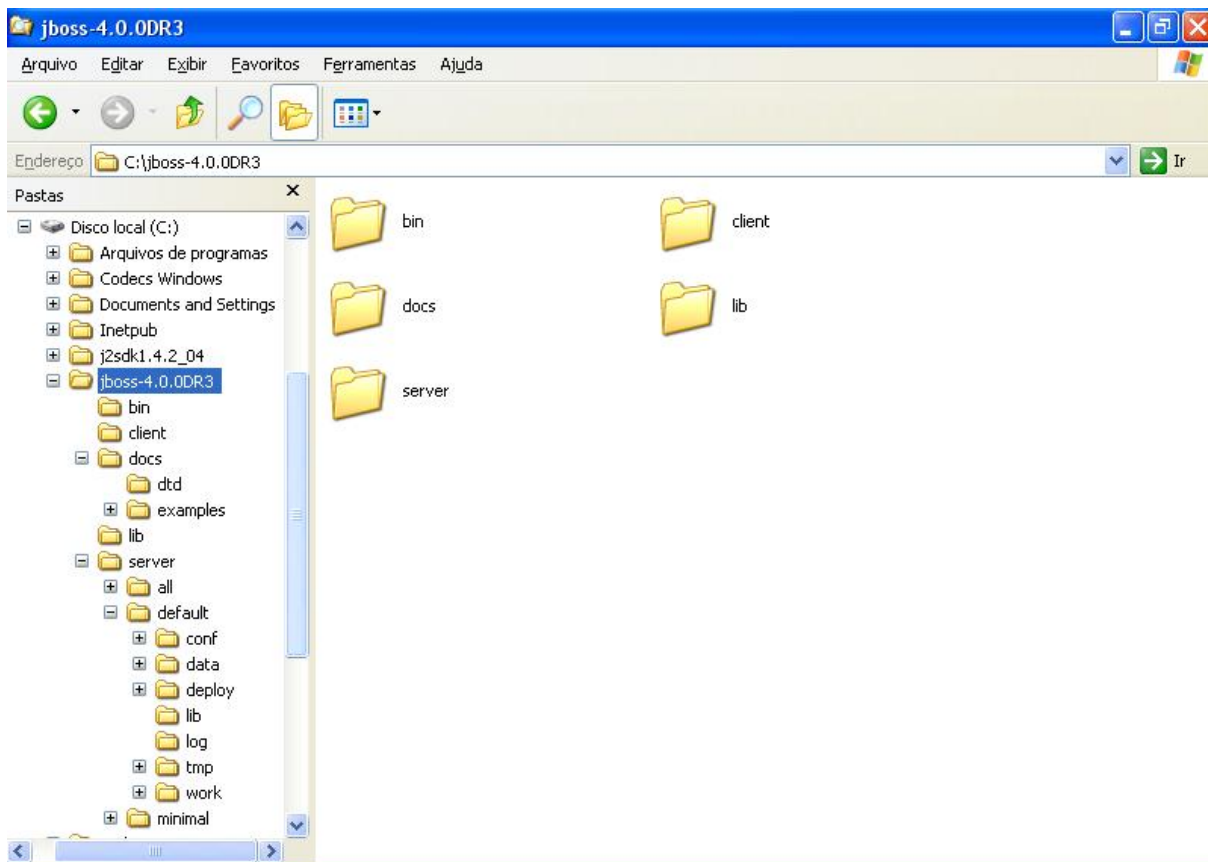


Figura 12. Estrutura de diretórios no *JBoss*

O diretório *JBOSS_DIST/server* contém um ou mais conjuntos de arquivo de configuração. A arquivo padrão de configuração fica situado no diretório *JBOSS_DIST/server/default*. *JBoss* permite mais de um conjunto de configuração executando de forma alternada no servidor.

Dentro do diretório *server*, há três configurações de exemplo: *all*, *default* e *minimal*, cada uma instala um conjunto diferente de serviços. Não surpreendentemente, a configuração *default* é usada se não for passado qualquer parâmetro ao *script* de execução.

A configuração *default* contém tudo o que você precisa executar *stand-alone J2EE server*. As outras duas são *minimal* - o mínimo exigido para iniciar o *JBoss*. Iniciam o serviço de *Log*, um servidor de *JNDI* e um *scanner* de desenvolvimento de *URL* para achar desenvolvimentos novos. Isto é necessário para usar *JMX/JBoss* para iniciar seus próprios serviços sem qualquer outro serviço *J2EE*, para esta configuração - não há *web container*, nenhum *EJB* ou *JMS*. A configuração *ALL* - inicia todos os serviços disponíveis. Isto inclui o

RMI/IIOP e agrupando serviços e o *deployer* de *web-services* que não está carregado na configuração *default*.

Instalado e configurado o *JBoss* conforme as necessidades de cada caso, basta agora ao desenvolvedor o papel de colocar em prática sua aplicação e utilizar toda as vantagens e facilidades oferecidas pelo *JBoss*.

Considerações finais deste capítulo

Neste capítulo foi abordado o requisito necessário para instalação do *JBoss* e onde podemos encontrá-lo para *download*, sugestões de configuração e sua estrutura com a descrição de alguns diretórios.

RESULTADOS OBTIDOS

Segundo as facilidades demonstradas anteriormente quanto a instalação e aquisição do *JBoss*, e o que o difere dos seus concorrentes, estas facilidades podem ser fundamentais na escolha de um servidor de aplicação e ainda mais distribuído pela licença *LGPL*.

Por ser *Open Source*, o caminho seguido pelo *JBoss* para distinção é: integra-se muito bem com uma série de ferramentas *Open Source* para a criação de ambientes de desenvolvimento de *software* bastante completa e robusta.

Segundo as especificações técnicas o *JBoss 4.X* inclui um *framework* para *Aspect Oriented Programming – AOP* (*JBOSS, 2004*) que permite aos desenvolvedores, com facilidade adicionar serviços como transações, persistência e replicação de *cache* a partir de objetos *Java* comuns, sem a necessidade de criar *EJBs*. O *framework AOP* cuida de todo o trabalho de transformar o objeto *Java* em um *EJB*, e libera o desenvolvedor para se concentrar na lógica de negócio das aplicações.

Quanto a sua instalação após os *downloads* dos requisitos necessários para instalação do *JBoss*, inicialmente o primeiro passo foi a instalação do *JSDK* que ocorreu sem problema algum com a execução do arquivo de instalação *j2sdk-1_4_2-nb-3_5_1-bin-windows.exe* onde, praticamente, é tudo automático, lembrando apenas que foram necessários a configuração da variável de ambiente *JAVA_HOME*, e adicionar no *path* do sistema operacional o diretório onde se encontra o *JSDK*.

Após este passo foi descompactado o arquivo *jboss-4.0.0DR3.zip* no *C:*, e através do *Prompt do Dos* no diretório *C:\jboss-4.0.0DR3\bin* a execução do comando *run.bat* mais a opção desejada para o tipo de servidor (*minimal, default, all*). Finalizando o seu carregamento pode-se chamar o *browser* de sua preferência e iniciar as telas de configuração do *JBoss* pelo endereço *http://127.0.0.1:8080/jmx-console*.

Sem dúvida tratando-se de um servidor de aplicação sua instalação é simples e rápida.

CONCLUSÃO

Através do estudo realizado sobre *JBoss* existem fatos importantes a serem citados como: os servidores de aplicação *J2EE* estarem ganhando um amplo impulso, devido à produtividade no desenvolvimento de aplicações corporativas distribuídas e facilidade que têm no aproveitamento de sistemas e bases de dados relacionais em novas aplicações. Estes servidores estão se tornando a infra-estrutura da nova geração de aplicações corporativas, seja para utilização intracorporação, seja em aplicações *B2B* e *B2C*.

Em um cenário onde os executivos de *TI* têm cada vez maiores restrições orçamentárias, além disso, enorme pressão por resultados, é uma alternativa para servidor de aplicação *Open Source* de grande qualidade. Assim como o *Linux*, o *Apache* e outros *softwares Open Source*, o *JBoss* está se tornando elemento essencial nas decisões de *TI* das grandes corporações.

Um grande trunfo do ponto de vista de sua arquitetura interna: o *JBoss* é baseado em uma arquitetura de *microkernel JMX*, onde todos os módulos que compõem o servidor, além das próprias aplicações, são componentes (*MBeans*) “plugados” ou substituídos dinamicamente, em *runtime*, sem a necessidade de paradas no servidor. Esta funcionalidade, que é chamada de “*hot deploy*”, dá uma grande flexibilidade e robustez ao servidor.

Para justificar os altos preços cobrados pelos concorrentes comerciais do *JBoss*, os desenvolvedores destes sistemas procuram se diferenciar oferecendo acoplado com o Servidor de Aplicação, um conjunto de ferramentas e outros *softwares* tais como: ambientes de desenvolvimento de aplicações, gerenciadores de conteúdo, portais e etc.

Mesmo sendo o *JBoss Open Source*, há uma dificuldade em relação a sua documentação que esta sim é paga. Outro incômodo é que atualmente somente está disponível em inglês, fato este que desencorajam muitos, pois há muitos termos técnicos, o que pode vir a gerar dúvidas sobre alguns aspectos.

Conclui-se mesmo com algumas dificuldades encontradas quanto à documentação escrita e *on-line (web)*, o *JBoss* cativa com respeito a rapidez no requisito de instalação, à qualidade de seus recursos, viabilidade, tecnologia utilizada e uma ampla visão quanto ao futuro dos produtos *Open Source*, onde se pode observar no próprio portal do *JBoss* na seção *news* a crescente procura e interesse de outras empresas e pessoas quanto à integração e a procura do conhecimento mais aprofundados sobre o *JBoss*.

REFERÊNCIAS

CARVILHE, J. L. V. A utilização de tecnologias web em sistemas de gerência corporativa. Curitiba: PUC-PR, 2000. (*Monografia apresentada no Curso de Especialização em Sistemas Distribuídos*).

HARNEDY, S. Web-based management for the enterprise. New Jersey: Prentice Hall, 1999.

LABOUREY, S.; BURKE, B., and The JBoss Group - JBoss Clustering. Atlanta Dez. 2002.

STARK, S. and The JBoss Group - JBoss Administration and Development Second Edition. Atlanta Nov. 2002.

LABOUREY, S.; BURKE, B. - JBoss 3.0 WorkBook for Enterprise JavaBeans 3^a Edition. Atlanta. 2002.

CARDELINI, V; COLAJANNI, M; YU, P. S. - Dinamic Load Balancing on Web-server Systems

BUNT, R. B; EAGER, D. L.; OSTER, G. M., and WILLIAMSON, C. L.; Achieving Load Balance and Effective Caching in Clustered Web Servers

TEO, Y. M.; AYANI, R - Comparison of Load Balancing Strategies on Cluster-based Web Servers

CARDELINI, V; COLAJANNI, M; YU, P. S. - Geographic Load Balancing for Scalable Distributed Web Systems

ROCHA, Helder da; Minicursos – Java . J523 – Tutorial JNDI, 2004

MIC99 - 1999, **Sun Microsystems**. Java management extensions white paper. Technical report, Palo Alto - CA, Junho 1999

PERENS -1997, **Bruce** - The Debian Free Software Guidelines – junho de 1997

SANTOS, **Carlos A. M. dos**, BSD e GPL, 2003.

LINKS

VISWANATHAN, Vivek; Load Balancing Web Applications, 2001

<http://www.onjava.com/pub/a/onjava/2001/09/26/load.html>

BURKE, B., LABOUREY, S.; Clustering with JBoss 3.0, 2002

<http://www.onjava.com/pub/a/onjava/2002/07/10/jboss.html>

BURKE, B., Clustering with JBoss/Jetty, 2001

<http://www.onjava.com/pub/a/onjava/2001/09/18/jboss.html>

SCHAEFER, A., Using JBoss Web Application Server, 2001

<http://www.onjava.com/pub/a/onjava/2001/07/16/jboss.html>

CARVILHE, José Luís, Bate Byte 100 Agosto/2000 – Java Management Extension, 2000

<http://www.pr.gov.br/batebyte/edicoes/2000/bb100/java.htm>

DOMINGUES, André Luís dos Santos - Extensible Markup Language – XML, 2003

<http://www.icmc.sc.usp.br/~alsd/icmc-usp-disciplina-hm-seminario-www-xml.html>

ROCHA, Helder da; Argo Navis Informática e Consultoria S/C Ltda, 2004

<http://www.argonavis.com.br>

Revista On-Line sobre Java

<http://www.javaworld.com>

SUN, 2004 – Sun Microsystems

<http://www.sun.com>

JBOSS, 1999 – JBoss :: Professional Open Source

<http://www.jboss.org>

Mundo OO, 2004 – Mundo OO

<http://www.mundooo.com.br>

W3C, 2004 – Word Wide Web Consortium

<http://www.w3.org>

POINTBASE, 2004 – The Point Base

<http://www.pointbase.com>, <http://www.hypersonicsql.com>

JAKARTA, 2004 – The Jakarta Site – Apache Tomcat

<http://jakarta.apache.org/tomcat>

JETTY, 2003 – Jetty Java HTTP Servlet Server

<http://jetty.mortbay.org/jetty/index.html>

IMASTER, 2001 – iMasters FFPA

<http://www.imasters.com.br/>

JBOSSSX, 2004 – The JBossSx Default Security Manager

<http://pipin.tmd.ns.ac.yu/extra/java2/libs/JBoss/ch09s09.html>

ROB, JOHNSON, - J2EE Design and Development - J2EE Overview, 2003

http://www.javacampinas.com.br/palestras/evento/JavaCampinas2003_J2EE.pdf