

**LIGA DE ENSINO DO RIO GRANDE DO NORTE
FACULDADE NATALENSE PARA O DESENVOLVIMENTO
DO RIO GRANDE DO NORTE**

DANIEL QUERINO OLÍMPIO

**ADMINISTRAÇÃO DE SERVIDOR DE APLICAÇÃO
JBoss APPLICATION SERVER**

**ADMINISTRAÇÃO DE SERVIDOR DE APLICAÇÃO
JBoss APPLICATION SERVER**

Disciplina do Curso de MBA
em Desenvolvimento de Sistemas
Corporativos da Faculdade Natalense
para o Desenvolvimento do Rio Grande do
Norte como requisito obrigatório para
obtenção do grau de Especialista em
Sistemas Corporativos.

Orientador: Prof. Gleydson Lima, MSc.

DANIEL QUERINO OLIMPIO

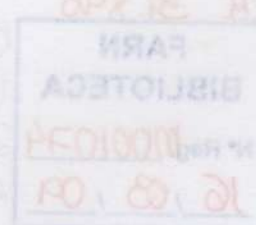
ADMINISTRAÇÃO DE SERVIDOR DE APLICAÇÃO JBOSS APPLICATION SERVER

ADMINISTRAÇÃO DE SERVIDOR DE APLICAÇÃO JBOSS APPLICATION SERVER

Monografia apresentada ao Curso de MBA em Desenvolvimento de Sistemas Corporativos da Faculdade Natalense para o Desenvolvimento do Rio Grande do Norte como requisito obrigatório para obtenção do grau de Especialista em Sistemas Corporativos.

Orientador: Prof. Gleydson Lima, MSc.

NATAL
2008



Catálogo na Publicação – Biblioteca da FARN
Setor de Processos Técnicos

Olimpio, Daniel Querino

Administração de servidor de aplicação JBOSS application server /
Daniel Querino Olimpio. – Natal: FARN, 2008.
53f.

Orientador: Gleydson Lima (Msc.)

Monografia (Especialização em Desenvolvimento de Sistemas
Corporativos) – Faculdade Natalense para o Desenvolvimento do Rio
Grande do Norte.

1. Desenvolvimento de Software – Monografia. 2. JAVA – Monografia.
3. Servidor de Aplicação – Monografia. 4. JBoss – Monografia. I. Título.

RN/FARN/BC

CDU 004.43



004.43
0392
2008
ex 01

NATAL
8008

DANIEL QUERINO OLIMPIO

Este trabalho tem como objetivos apresentar o servidor de aplicação JBoss AS, detalhando alguns componentes de sua arquitetura, como JMX, JNDI, JBoss Transações, JBoss Segurança, JBoss Clustering, abordando desde sua instalação, configuração de sua estrutura, definição de parâmetros de localização, enfocando em alguns pontos-chaves relacionados principalmente a sua administração, sendo também apresentadas algumas ferramentas que utilizam o JBoss como servidor de aplicação. Também será apresentada uma breve visão geral sobre a arquitetura da plataforma J2EE.

ADMINISTRAÇÃO DE SERVIDOR DE APLICAÇÃO JBoss APPLICATION SERVER

Palavras-chave: Java EE, JBoss, Servidor de Aplicação

Monografia apresentada ao Curso de MBA em Desenvolvimento de Sistemas Corporativos da Faculdade Natalense para o Desenvolvimento do Rio Grande do Norte, aprovada em _____ de _____ de _____ pela banca examinadora composta pelos seguintes membros:

Gleydson Lima, MSc.
Orientador

RESUMO

Este trabalho tem como objetivos apresentar o servidor de aplicação JBoss AS, detalhando alguns componentes de sua arquitetura, como JMX, JNDI, JBoss Transações, JBoss Segurança, JBoss Clustering, abordando desde sua instalação, configuração de sua estrutura, definição de parâmetros de inicialização, enfocando em alguns pontos chaves relacionados principalmente a sua administração, sendo também apresentadas algumas ferramentas que utilizam o JBoss como servidor de aplicação. Também será apresentada uma breve visão geral sobre a arquitetura da plataforma J2EE, não aprofundando em suas características e funcionalidades.

Keywords: Java EE, JBoss, Application Server

Palavras-chave: Java EE. JBoss. Servidor de Aplicação

LISTA ABSTRACT TUPAS

This work aims to present the application server, JBoss AS, detailing some components of its architecture, such as JMX, JNDI, JBoss Transactions, JBoss Security, JBoss Clustering, addressing since its installation, configuration of its structure, definition of parameters for initialization, focusing on some key points relating mainly to its administration, and also presented some tools that use JBoss as a server application. It will be a brief overview on the architecture of the J2EE platform, no depth in its features and functionalities.

Keywords: Java EE. JBoss. Application Server

JAR	Java Archive
Java SE	Java Platform, Standard Edition
JDBC	Java Database Connectivity
JMS	Java Message Service
JMX	Java Management Extensions
JNDI	Java Naming and Directory Interface
JRMP	Java Remote Method Protocol
JSP	Java Server Pages
JTA	Java Transaction API
JTS	Java Transaction Service
JVM	Java Virtual Machine
RMI	Remote Method Invocation
SOA	Service-Oriented Architecture
SQL	Structured Query Language
TCP	Transmission Control Protocol
TPC	Transaction Propagation Context
UDP	User Datagram Protocol
VM	Virtual Machine
XML	Extensible Markup Language

LISTA DE ABREVIATURAS

API	Application Programming Interface	
BD	Banco de Dados	
DNS	Domain Name System	12
EIS	Enterprise Information System	15
EJB	Enterprise Java Beans	19
HTML	Hyper Text Markup Language	
HTTP	Hypertext Transfer Protocol	
JAR	Java Archive	
Java EE	Java 2 Platform, Enterprise Edition	21
Java SE	Java Platform, Standard Edition	24
JDBC	Java Database Connectivity	26
JMS	Java Message Service	27
JMX	Java Management Extensions	27
JNDI	Java Naming and Directory Interface	29
JRMP	Java Remote Method Protocol	36
JSP	Java Server Pages	42
JTA	Java Transaction API	43
JTS	Java Transaction Service	45
JVM	Java Virtual Machine	
RMI	Remote Method Invocation	
SOA	Service-Oriented Architecture	
SQL	Structured Query Language	
TCP	Transmission Control Protocol	
TPC	Transaction Propagation Context	
UDP	User Datagram Protocol	
VM	Virtual Machine	
XML	Extensible Markup Language	

LISTA DE FIGURAS

FIGURA 1 Camadas da plataforma Java EE	12
FIGURA 2 Comunicação entre as camadas	15
FIGURA 3 Visão da estrutura de diretórios de instalação do JBoss	19
FIGURA 4 Visão expandida dos diretórios conf e deploy do conjunto de arquivos de configuração padrão.....	21
FIGURA 5 Principais componentes da carga de classes do JBoss	24
FIGURA 6 Visualização do MBean LoaderRepository	26
FIGURA 7 Integração JMX e os componentes padrão do JBoss.....	27
FIGURA 8 Relacionamentos entre os componentes da arquitetura JMX.....	28
FIGURA 9 Principais componentes da arquitetura JBossNS	36
FIGURA 10 Visualização da aplicação JMX Console	42
FIGURA 11 Visualização da aplicação WEB Console	43
FIGURA 12 Exemplo de instâncias de servidores JBoss divididas em partições	46

LISTA DE QUADROS

1 INTRODUÇÃO	10
QUADRO 1 Descrição da estrutura de diretórios do JBoss	20
QUADRO 2 Parâmetros utilizados na inicialização do JBoss	22
QUADRO 3 Portas de configuração padrão.....	40
QUADRO 4 Portas adicionais na configuração de servidor do tipo all.....	41
QUADRO 5 Definição do Clustering MBean	47
QUADRO 6 Parâmetros de configuração do arquivo cluster-service.xml	48
QUADRO 7 Conjunto de protocolos utilizados na configuração da partição.....	49
QUADRO 8 Conteúdo do arquivo farm-service.xml	50
QUADRO 9 Parâmetros do arquivo de configuração farm-service.xml.....	50
2.3 JAVA EE CONTÊINERES	14
2.3.1 Contêiner Client	15
2.3.2 Contêiner Applet	15
2.3.3 Contêiner Web	15
2.3.4 Contêiner EJB	16
2.3.5 Java Enterprise Edition 5	16
3 JBOSS APPLICATION SERVER	18
3.1 INSTALAÇÃO	18
3.1.1 Estrutura de Diretórios	18
3.1.2 Configuração Padrão do Servidor	21
3.1.3 Parâmetros de Inicialização	21
3.2 JAVA MANAGEMENT EXTENSIONS (JMX)	23
3.2.1 Arquitetura ClassLoader	23
3.2.2 Nível de instrumentação	23
3.2.3 Nível de Agentes	29
3.2.4 Nível de Serviços Distribuídos	29
3.2.5 Componentes do JMX	30
3.2.5.1 MBeans	31

SUMÁRIO

1 INTRODUÇÃO	10
2 JAVA EE	11
2.1 VISÃO GERAL	11
2.2 ARQUITETURA DA PLATAFORMA JAVA EE	11
2.2.1 Componentes Cliente	12
2.2.2 Componentes Web	13
2.2.3 Componentes Negócios	13
2.2.4 Componentes Enterprise Information System (EIS)	14
2.3 JAVA EE CONTÊINERES	14
2.3.1 Contêiner Cliente	15
2.3.2 Contêiner Applet	15
2.3.3 Contêiner Web	16
2.3.4 Contêiner EJB	16
2.3.5 Java Enterprise Edition 5	16
3 JBOSS APPLICATION SERVER	18
3.1 INSTALAÇÃO	18
3.1.1 Estrutura de Diretórios	18
3.1.2 Configuração Padrão do Servidor	21
3.1.3 Parâmetros de Inicialização	21
3.2 JAVA MANAGEMENT EXTENSIONS (JMX)	23
3.2.1 Arquitetura ClassLoader	23
3.2.2 Nível de Instrumentação	28
3.2.3 Nível de Agentes	29
3.2.4 Nível de Serviços Distribuídos	29
3.2.5 Componentes do JMX	30
3.2.5.1 MBeans	31

3.2.5.2 Elementos do modelo de notificações.....	32
3.2.5.3 Classe de metadados do MBean	32
3.2.5.4 O servidor de MBeans.....	32
3.2.5.5 Serviços de agente.....	33
3.3 JBOSS JNDI	34
3.3.1 Tratamento de Nomes no JNDI.....	35
3.3.2 Contextos	35
3.3.3 Arquitetura JBossNS	35
3.4 JBOSS TRANSAÇÕES.....	37
3.4.1 Componentes de uma Transação Distribuída	37
3.4.2 Detalhes Internos das Transações no JBoss	37
3.4.3 Gerenciador de Transações Padrão	38
3.5 JBOSS SEGURANÇA.....	39
3.5.1 Modelo de Segurança do JBoss.....	39
3.5.2 Configurando o JBoss para ser usada atrás de Firewall.....	40
3.5.3 Protegendo o JBoss.....	41
3.5.3.1 Serviço jmx-console.war	41
3.5.3.2 Serviço web-console.war	42
3.5.3.3 Serviço http-invoker.sar.....	43
3.5.3.4 Serviço jms-invoker-adaptor-server.sar.....	44
3.6 JBOSS CLUSTERING	44
3.6.1 Características	45
3.6.2 Partição	45
3.6.3 Configurando um Cluster	47
3.6.4 Farming Deployment.....	49
4 CONCLUSÃO	51
REFERÊNCIAS	52

1 INTRODUÇÃO

A utilização de servidores de aplicação vem se tornando cada vez mais freqüente e recomendado para empresas disponibilizarem seus sistemas que utilizam a plataforma J2EE. Como exemplo de alguns recursos que esses servidores oferecem, temos: alta disponibilidade; alta capacidade para acesso simultâneo nos contêineres WEB e EJB; facilidades de configuração e utilização de banco de dados; facilidade de alteração, publicação e disponibilização de sistemas em tempo real; segurança; balanceamento de carga e tolerância a falhas. Alguns dos servidores de aplicações de código aberto J2EE mais populares do mercado, são: JBoss, JOnAS e Geronimo (projeto de servidor de aplicação da Apache Foundation Group).

O JBoss é um servidor de aplicações de código aberto tendo como diferencial uma excelente integração com uma série de ferramentas open source para a criação de ambientes de desenvolvimento de software bastante completo e robusto. Por exemplo: o IDE Eclipse, o gerenciador de versão de software CVS, etc.

Ele também é um servidor de aplicações certificado com o padrão J2EE 1.4. Sua arquitetura de servidor altamente flexível e fácil de usar tornou o JBoss uma excelente opção para usuários que estão em busca de uma plataforma de middleware personalizável.

2.2 ARQUITETURA DA PLATAFORMA JAVA EE

De acordo com Sun (2006) a plataforma Java EE é dividida em quatro camadas que proporcionam escalabilidade para as aplicações. As camadas são Cliente, Web, Negócios e Enterprise Information System (EIS). Geralmente, considera-se que a plataforma é dividida em três camadas, pois são distribuídas em três máquinas distintas: a máquina cliente, a máquina servidor Java EE e a base de dados ou sistemas legados.

2 JAVA 2 EE

Nesta seção será abordada a arquitetura Java 2 Enterprise Edition (Java EE) detalhando como essa é dividida dando uma visão geral de cada um de seus componentes.

Primeiramente, serão abordados os componentes que compõem a arquitetura e como se relacionam entre si. Depois cada um desses componentes é detalhado deixando mais claro o papel de cada um na arquitetura.

2.1 VISÃO GERAL

A plataforma Java 2 Enterprise Edition (J2EE) é um ambiente para desenvolvimento e execução de aplicações servidoras, com capacidade de suporte ao desenvolvimento de aplicações robustas e escaláveis. Consistindo em um conjunto de serviços, interfaces de programação de aplicação (APIs) e protocolos, que oferecem a funcionalidade para o desenvolvimento de aplicações de multicamadas, baseadas na Web.

2.2 ARQUITETURA DA PLATAFORMA JAVA EE

De acordo com Sun (2006) a plataforma Java EE é dividida em quatro camadas que proporcionam escalabilidade para as aplicações. As camadas são Cliente, Web, Negócios e Enterprise Information System (EIS). Geralmente, considera-se que a plataforma é dividida em três camadas, pois são distribuídas em três máquinas distintas: a máquina cliente, a máquina servidora Java EE e a base de dados ou sistemas legados.

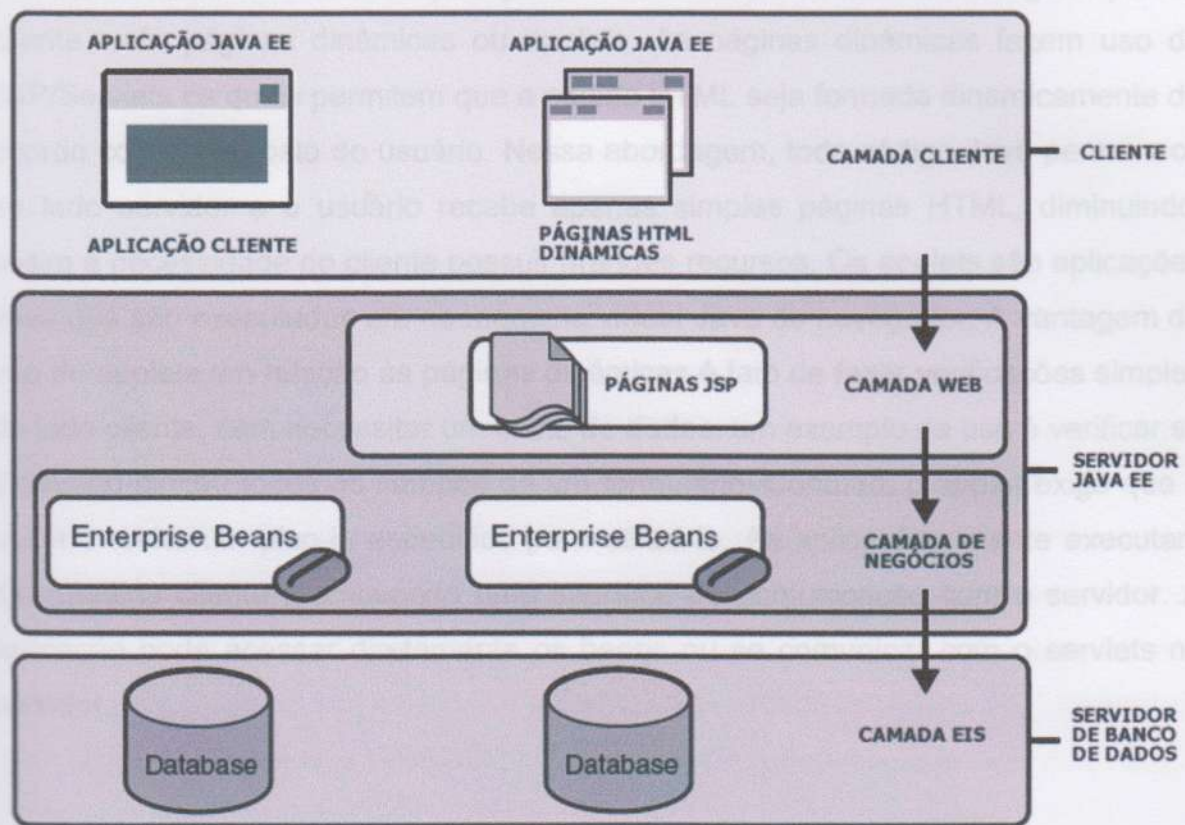


Figura 1 Camadas da plataforma Java EE
Fonte: Sun (2006)

Segundo Sun (2006) cada camada é composta por um conjunto de componentes. Componentes Java EE são unidades funcionais de software auto-contidas que são montadas em uma aplicação Java EE com suas classes e arquivos relacionados e se comunicam com outros componentes. Uma forma de comunicação opcional entre as camadas é o componente JavaBeans que pode ser uma classe qualquer Java que segue algumas convenções, e objetiva ser um componente de fácil reuso e composição com outras aplicações.

2.2.1 Componentes Cliente

Os componentes clientes são localizados no lado do cliente e proporcionam a interface de comunicação entre o usuário e o servidor. Os componentes clientes são

divididos em clientes web e aplicações clientes. Existem duas abordagens para o cliente web: páginas dinâmicas ou applets. As páginas dinâmicas fazem uso do JSP/Servlets os quais permitem que a página HTML seja formada dinamicamente de acordo com a resposta do usuário. Nessa abordagem, todo código Java permanece no lado servidor e o usuário recebe apenas simples páginas HTML, diminuindo, assim a necessidade do cliente possuir grandes recursos. Os applets são aplicações Java que são executadas em na máquina virtual Java do navegador. A vantagem do uso de applets em relação às páginas dinâmicas é fato de fazer verificações simples do lado cliente, sem necessitar um envio de dados, um exemplo de uso é verificar se o usuário digitou todos os campos de um formulário. Contudo, o applet exige que o usuário tenha um plug-in específico para utilizá-lo. As aplicações cliente executam na máquina cliente promovendo uma interface de comunicação com o servidor. A aplicação pode acessar diretamente os beans ou se comunicar com o servlets no servidor.

2.2.2 Componentes Web

Os componentes web Java EE estão localizados no servidor podem ser páginas JSP, Servlets. Esses componentes são tratados pelo contêiner web, o qual prove funcionalidades, como por exemplo, segurança. Outros elementos utilizados juntamente com os componentes web como applets e páginas HTML não são considerados componentes web. Os servlets são classes Java que produzem requisições dinamicamente e constroem respostas. Por sua vez, páginas JSP são documentos baseados em texto com o funcionamento similar aos servlets, porém permitem o uso de elementos estáticos.

2.2.3 Camada de Negócios

Define a parte lógica da aplicação localizado no servidor de aplicações Java EE. Nessa camada encontram-se os Enterprise JavaBeans (EJB). O contêiner EJB é

responsável pelo tratamento desses componentes e fornecem serviços cuja função é fazer a parte funcional do código, como por exemplo, as transações, deixando assim a cargo do programador só a parte lógica.

2.2.4 Enterprise Information System (EIS)

De acordo com Sun (2006) nesta camada encontram-se programas EIS como sistemas legados, banco de dados. A conexão com a camada EIS normalmente é feita através de conectores fornecidos pelo fabricante do componente na camada EIS. Um exemplo de conector é o JDBC cuja função é oferecer conexão ao banco de dados de tal maneira que independente do fornecedor usado, a modo de acesso é o mesmo. Quando não existe um conector, aconselha-se utilizar uma padrão de projeto que evita acessos diretos ao componente EIS de diversos pontos do código.

Figura 2 Comunicação entre as camadas
Fonte: Sun (2006)

2.3 JAVA EE CONTÊINERES

Segundo Sun (2006) os componentes das diversas camadas do Java EE são gerenciados pelos contêineres. Contêineres são as interfaces entre componentes e a funcionalidade baixo nível de uma plataforma específica que suporta o componente. Esses contêineres fornecem serviços que facilitam o desenvolvimento de aplicações. Dentre os serviços fornecidos, destacam-se como a transação que permite utilizar uma transação a qual engloba diversos métodos, segurança que permite definir acesso aos métodos através da definição de autorizações e o JNDI que prove uma interface unificada para o serviço de nomes. Existem diversos tipos de contêineres, como: contêiner de aplicação, o contêiner applet, o contêiner web e o contêiner EJB.

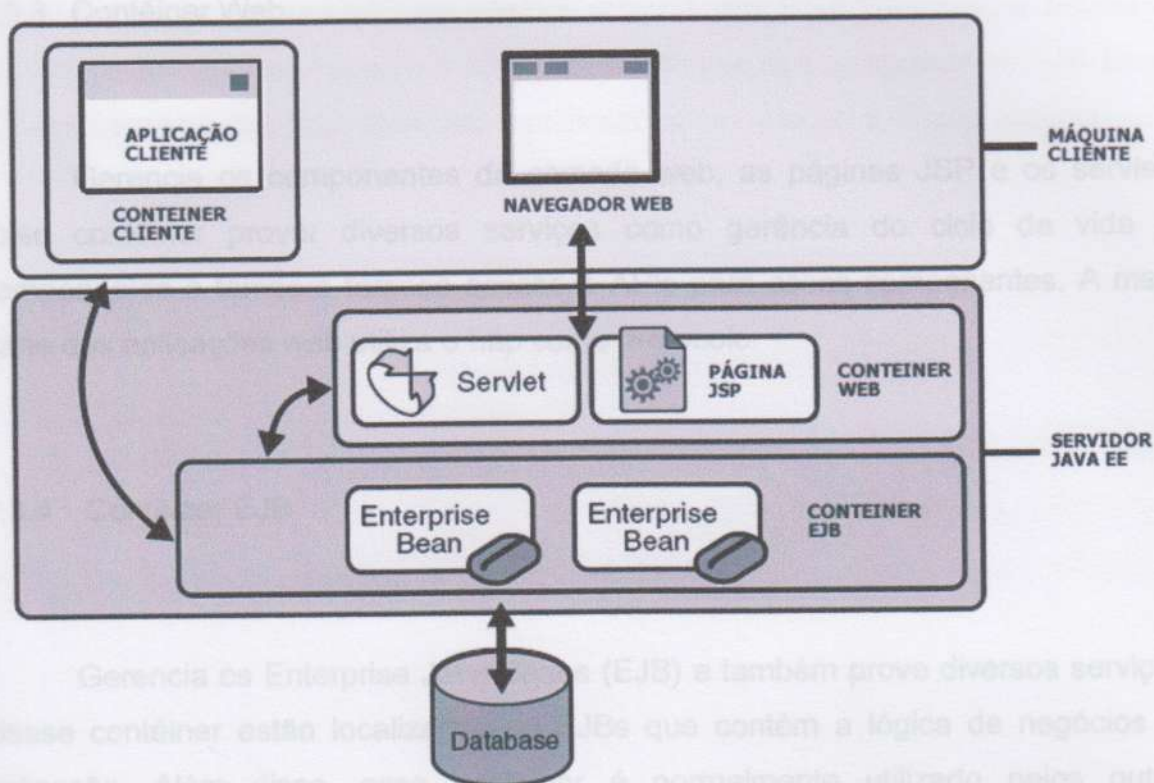


Figura 2 Comunicação entre as camadas
Fonte: Sun (2006)

2.3.1 Contêiner Cliente

Localizado na camada cliente, gerencia as aplicações do lado cliente. O contêiner pode acessar diretamente ao contêiner EJB ou ao contêiner web.

2.3.2 Contêiner Applet

Localizado no lado cliente é composto por um plug-in Java e uma página HTML executados em conjunto. Exige dessa forma a existência do plug-in Java do lado cliente, tornando a aplicação um pouco mais pesada.

2.3.3 Contêiner Web

Esses são um melhor suporte aos Web Services, a adição da tecnologia JavaServer Pages e a simplificação no uso dos componentes EJB. Essas últimas foram as que mais sofreram modificações em relação à versão anterior.

Gerencia os componentes da camada web, as páginas JSP e os servlets. Esse contêiner prover diversos serviços como gerência do ciclo de vida de componentes e também fornece acesso à APIs para esses componentes. A maior parte das aplicações web utiliza o http como protocolo.

2.3.4 Contêiner EJB

Gerencia os Enterprise Java Beans (EJB) e também prove diversos serviços. Nesse contêiner estão localizados os EJBs que contêm a lógica de negócios da aplicação. Além disso, esse contêiner é normalmente utilizado pelos outros contêineres para acesso à camada EIS.

2.3.5 Java Enterprise Edition versão 5

Esta versão apresenta como principais vantagens o aumento da conveniência, a melhoria na performance e redução no tempo de desenvolvimento, sendo essas originadas por seis fatores.

Um dos fatores que contribuem para essa melhoria é o fim da necessidade de utilizar descritores XML, que passam a ser opcionais. O suporte aos descritores foi mantido para garantir compatibilidade entre as versões e também para que fosse possível mudar as configurações sem alterar o código fonte.

O segundo fator é o uso de mais valores padrões, diminuindo o número de configurações necessárias para desenvolver uma aplicação. Assim, o desenvolvedor só define o que fugir da definição padrão.

O terceiro fator é a definição de uma nova API de persistência menos complexa que a da versão anterior. Além disso, com a nova API a parte de persistência pode ser utilizada também por aplicações Java SE.

Os outros fatores são um melhor suporte aos Web Services, a adição da tecnologia JavaServer Pages e a simplificação no uso dos componentes EJB. Esses últimos foram os que mais sofreram modificações em relação à versão anterior.

O JBoss Application Server (JBoss AS) foi lançado pela primeira vez em 1999 e é um servidor de aplicações Java 2 Enterprise Edition (J2EE) pronto para produção. Ele foi baseado na linha JBoss 3.2 de servidores de aplicações de código fonte aberto em conformidade de padrões e diversas melhorias adicionais no recursos.

De acordo com Fleury, Stark e Norman (2005, p. 3) foi certificado oficialmente como estando totalmente conforme a especificação J2EE 1.4. Sendo a versão 4.0 o primeiro servidor de aplicações J2EE 1.4 pronto para produção, permitindo que os desenvolvedores reutilizem seguramente componentes J2EE (como os Enterprise Java Beans (EJBs)) em diferentes servidores de aplicações. A certificação torna o JBoss uma alternativa segura tanto para usuários do JBoss quanto para usuários de outros servidores de aplicação.

Fornecer suporte total ao J2EE Web Services e à Service-Oriented Architecture (SOA).

Oferece suporte a programação orientada a aspectos (AOP, aspect-oriented programming) para o desenvolvimento de middleware.

Integra-se totalmente com o Hibernate, a estrutura de persistência de objetos de maior popularidade atualmente.

2.1 INSTALAÇÃO

2.1.1 Estrutura de Diretórios

A instalação de versão de distribuição do JBoss cria um diretório contendo os arquivos de inicialização do servidor, arquivos JAR, definições de configuração de servidor e diretórios de trabalho.

3 JBOSS APPLICATION SERVER

O JBoss Application Server (JBoss AS) foi lançado pela primeira vez em 1999 e é um servidor de aplicações Java 2 Enterprise Edition (J2EE) pronto para produção. Ele foi baseado na linha JBoss 3.2 de servidores de aplicações de código fonte aberto em conformidade de padrões e diversas melhorias adicionais no recursos.

De acordo com Fleury, Stark e Norman (2005, p. 3) foi certificado oficialmente como estando totalmente conforme a especificação J2EE 1.4. Sendo a versão 4.0 o primeiro servidor de aplicações J2EE 1.4 pronto para produção, permitindo que os desenvolvedores reutilizem seguramente componentes J2EE (como os Enterprise Java Beans [EJBs]) em diferentes servidores de aplicações. A certificação torna o JBoss uma alternativa segura tanto para usuários do JBoss quanto para usuários de outros servidores de aplicação.

Fornece suporte total ao J2EE Web Services e à Service-Oriented Architecture (SOA).

Oferece suporte a programação orientada a aspectos (AOP, aspect-oriented programming) para o desenvolvimento de middleware.

Integra-se totalmente com o Hibernate, a estrutura de persistência de objetos de maior popularidade atualmente.

3.1 INSTALAÇÃO

3.1.1 Estrutura de Diretórios

A instalação da versão de distribuição do JBoss criará um diretório contendo os scripts de inicialização do servidor, arquivos JAR, definições de configuração de servidor e diretórios de trabalho.

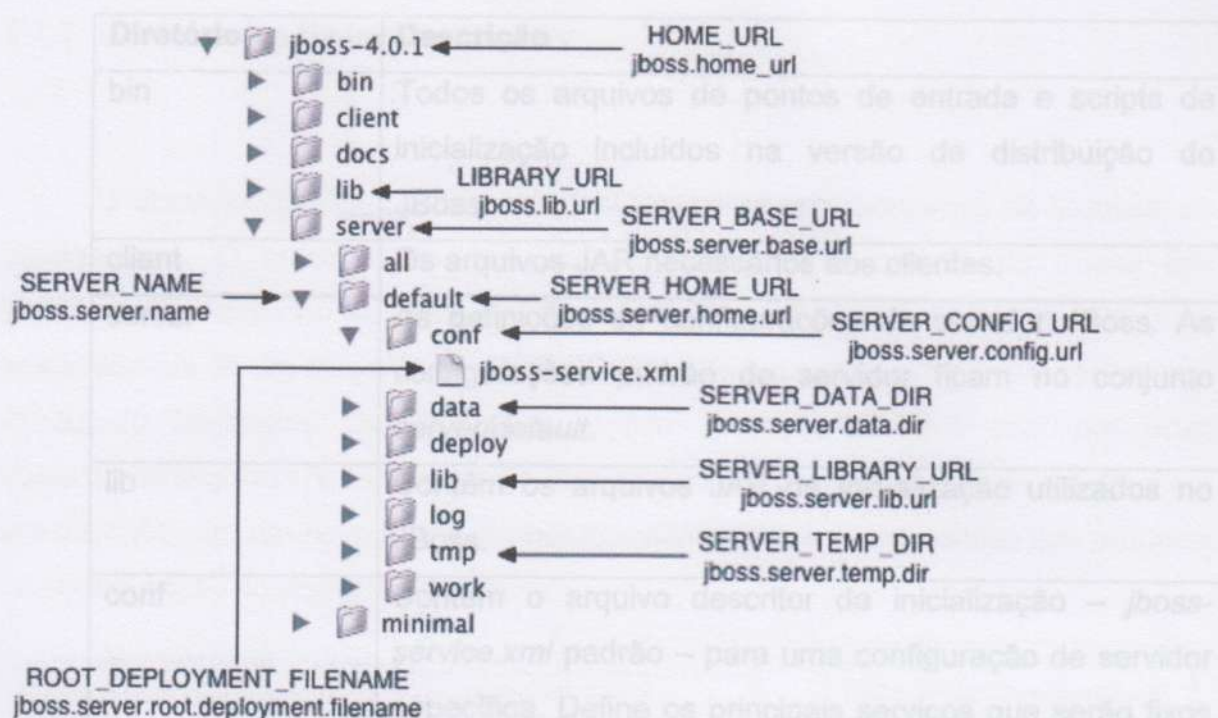


Figura 3 Visão da estrutura de diretórios de instalação do JBoss
Fonte: JBoss (2006)

data	Está disponível para ser usado pelos serviços que desejarem armazenar conteúdo no sistema de arquivos.
deploy	É o local padrão em que o serviço de implantação automática procura conteúdo de implantação.
lib	É o local padrão referenciado pelo descritor de inicialização. Todos os arquivos JAR desse diretório são carregados no caminho de classe compartilhado.
log	É o diretório padrão em que o serviço de registro de inicialização insere seus arquivos de log. Pode ser substituído através do arquivo de configuração <code>conf/log4j.xml</code> .
tmp	É o diretório onde as implantações são copiadas para uso local.

Quadro 1 Descrição de estrutura de diretórios

Fonte: JBoss (2006)

Diretório	Descrição
bin	Todos os arquivos de pontos de entrada e scripts de inicialização incluídos na versão de distribuição do JBoss.
client	Os arquivos JAR necessários aos clientes.
server	As definições de configurações do servidor JBoss. As configurações padrão do servidor ficam no conjunto <i>server/default</i> .
lib	Contém os arquivos JAR de inicialização utilizados no JBoss.
conf	Contém o arquivo descritor da inicialização – <i>jboss-service.xml</i> padrão – para uma configuração de servidor específica. Define os principais serviços que serão fixos durante a existência do servidor.
data	Está disponível para ser usado pelos serviços que desejarem armazenar conteúdo no sistema de arquivos.
deploy	É o local padrão em que o serviço de implantação automática procura conteúdo de implantação dinâmica.
lib	É o local padrão referenciado pelo descritor de inicialização. Todos os arquivos JAR desse diretório são carregados no caminho de classe compartilhado.
log	É o diretório padrão em que o serviço de registro de inicialização insere seus arquivos de log. Pode ser sobreposto através do arquivo de configuração <i>conf/log4j.xml</i> .
tmp	É o diretório onde as implantações são copiadas para uso local.

Quadro 1 Descrição da estrutura de diretórios

Fonte: JBoss (2006)

3.1.2 Configuração Padrão do Servidor

O diretório *JBOSS_DIST/server* contém um ou mais conjuntos de arquivos de configuração. O conjunto de arquivos de configuração padrão do JBoss fica localizado no diretório *JBOSS_DIST/server/default*. O JBoss permitirá que você adicione mais de um conjunto de configurações, de forma que um servidor possa ser executado facilmente usando configurações alternativas. Para criar um novo conjunto de arquivos de configuração faça a cópia do conjunto de arquivo *default* em um diretório com um novo nome, e posteriormente efetue as alterações dos arquivos de configuração conforme desejado.

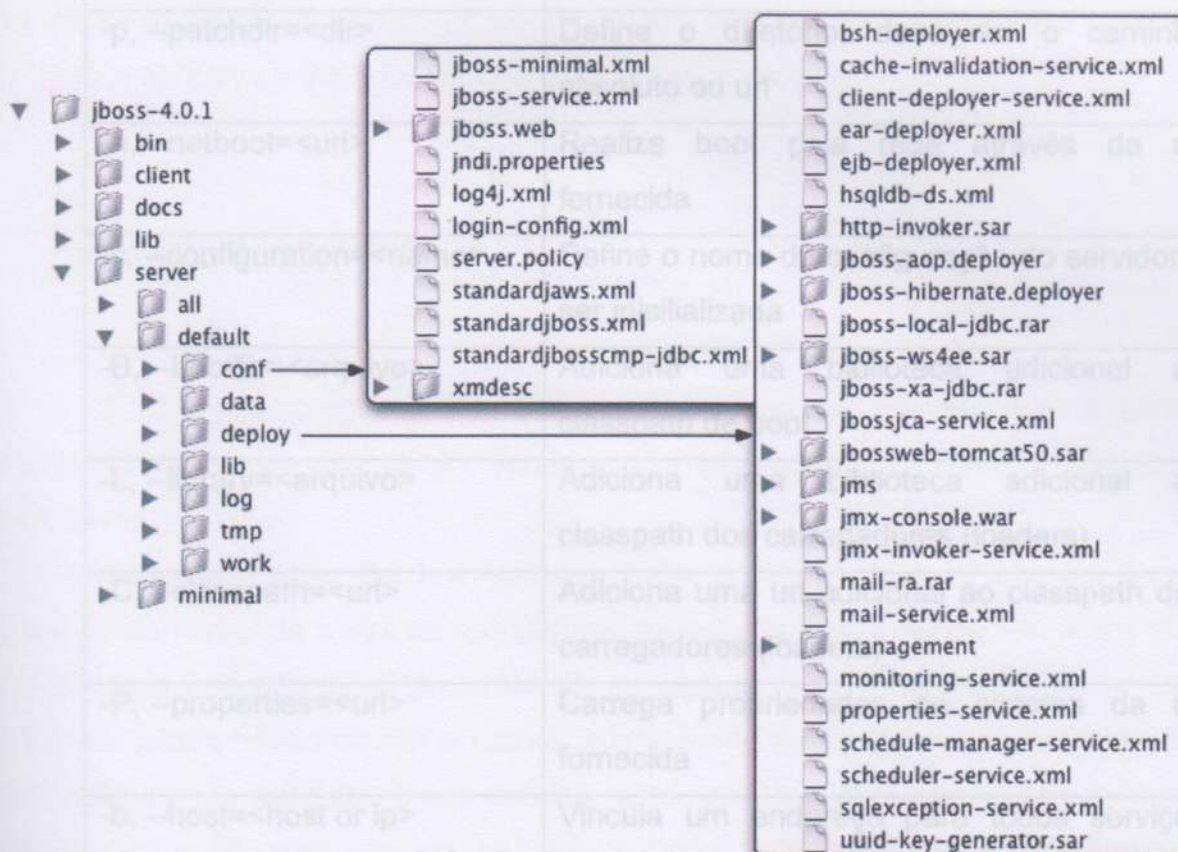


Figura 4 Visão expandida dos diretórios *conf* e *deploy* do conjunto de arquivos de configuração padrão.

Fonte: JBoss (2006)

3.1.3 Parâmetros de Inicialização

Ao iniciar o servidor de aplicação JBoss é possível passar algumas parâmetros a serem utilizados na configuração durante a inicialização

Parâmetro	Descrição
-h	Mostra mensagem de ajuda
-V	Mostra informações sobre a versão
--	Finaliza o processamento das opções
-D<nome>[=<valor>]	Define o valor de uma propriedade de sistema.
-d, --bootdir=<dir>	Define o diretório de boot; deve ser o caminho absoluto ou url
-p, --patchdir=<dir>	Define o diretório; deve ser o caminho absoluto ou url
-n, --netboot=<url>	Realiza boot pela rede através da url fornecida
-c, --configuration=<name>	Define o nome da configuração do servidor a ser inicializada
-B, --bootlib=<arquivo>	Adiciona uma biblioteca adicional ao classpath de boot
-L, --library=<arquivo>	Adiciona uma biblioteca adicional ao classpath dos carregadores (loaders)
-C, --classpath=<url>	Adiciona uma url adicional ao classpath dos carregadores (loaders)
-P, --properties=<url>	Carrega propriedades de sistema da url fornecida
-b, --host=<host or ip>	Vincula um endereço para todos serviços JBoss
-g, --partition=<nome>	Define o nome da partição (valor padrão=DefaultDomain)
-u, --udp=<ip>	Endereço UDP Multicast

Quadro 2 Parâmetros utilizados na inicialização do JBoss

Fonte: JBoss (2006)

De acordo com JBoss (2006) o servidor e o contêiner JBoss são totalmente implementados com o uso de plug-ins baseados em componentes. O esforço de modularização tem suporte no uso do JMX, a API Java Management Extensions. Com o uso do JMX, interfaces padrão da indústria ajudam a gerenciar tanto os componentes do servidor JBoss quanto os aplicativos implantados nele. Facilidade de uso continua sendo a prioridade numero um, e a arquitetura do servidor JBoss define um novo padrão para o projeto modular de plug-ins, assim como para a facilidade de gerenciamento de servidores e aplicativos.

Esse alto nível de modularidade beneficia os desenvolvedores de aplicativos de várias maneiras. O código já reduzido pode ser diminuído ainda mais para dar suporte a aplicativos que tenham que ocupar pouco espaço.

3.2.1 Arquitetura ClassLoader

O JBoss emprega uma arquitetura de carga de classes que facilita o compartilhamento de classes em unidades de implantação automática de serviços e aplicativos.

Conforme Fleury, Stark e Norman (2005, p.29) a carga de classes é uma parte fundamental de todas as arquiteturas de servidor, os serviços arbitrários e as classes que lhes dão suporte devem ser carregadas na estrutura do servidor. Isso pode ser problemático devido a natureza fortemente tipada de Java. A maioria dos desenvolvedores sabe que o tipo de uma classe em Java está relacionado ao nome totalmente qualificado da classe. No entanto, o tipo também depende da *java.lang.ClassLoader* que é usada para definir essa classe. Essa qualificação de tipo adicional é necessária para assegurar que ambientes nos quais classes possam ser carregadas a partir de locais arbitrários tenham consciência de tipos.

Porém em um ambiente dinâmico como o de um servidor de aplicações, como o JBoss, com seu suporte a implantação automática, exceções de conversão de classe, erros de vinculação e erros de acesso inválido podem ocorrer de maneiras não vistas em contextos estáticos de classes.

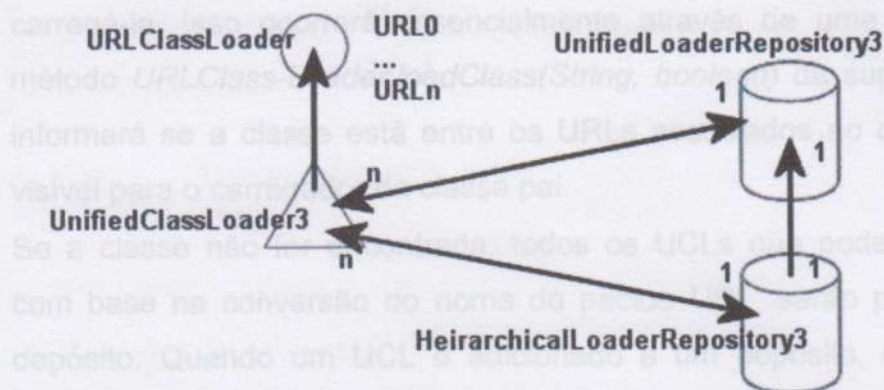


Figura 5 Principais componentes da carga de classes do JBoss.
Fonte: JBoss (2006)

Segundo JBoss (2006) o componente central da arquitetura de carga de classes do JBoss é o carregador de classes (UCL) *org.jboss.mx.loading.UnifiedClassLoader3*. Trata-se de uma extensão do carregador padrão java.net.URLClassLoader que sobrepõe o modelo padrão de delegação pai para usar um depósito compartilhado de classes e recursos. Esse depósito compartilhado *org.mx.loading.UnifiedLoaderRepository3*. Cada UCL está associado a apenas um *UnifiedLoaderRepository3* e normalmente tem muitos UCLs. Um UCL pode ter vários URLs associados a ele para a carga de classes e recursos. Os implantadores usam o UCL de implantação de nível superior como um carregador de classes compartilhado, e todos os arquivos de implantação são atribuídos a esse carregador.

Quando um UCL é solicitado a carregar uma classe, primeiro ele examina o cachê do depósito ao qual está associado para ver se a classe já foi carregada. O UCL só é carregada no depósito se a classe não existir nesse local. Por padrão, um único objeto *UnifiedLoaderRepository3* é compartilhado por todas as instâncias do UCL. Isso significa que os UCLs formam um único espaço de nome não-hierárquico de carregadores de classe. Fleury, Stark e Norman (2005, p.40) definem a sequência completa e as etapas que ocorrem quando um método *UnifiedLoaderRepository3.loadClass(String, boolean)* é chamado:

- Verifica o cache de classes *UnifiedLoaderRepository* associadas a *UnifiedClassLoader3*. Se a classe for encontrada no cache, será retornada.

- Se a classe não for encontrada, indagará ao *UnifiedClassLoader3* se pode carregá-la. Isso ocorrerá essencialmente através de uma chamada ao método *URLClass-Loader.loadClass(String, boolean)* da superclasse que informará se a classe está entre os URLs associados ao carregador ou visível para o carregador da classe pai.
- Se a classe não for encontrada, todos os UCLs que podem fornecê-la, com base na conversão do nome do pacote UCL, serão procurados no depósito. Quando um UCL é adicionado a um depósito, é criada uma associação entre os nomes de pacotes disponíveis nos URLs associados a esse UCL, e uma conversão dos nomes em UCLs de classes de pacote é atualizada. Isso permite uma determinação rápida de que UCLs podem carregar a classe. Em seguida, os UCLs são consultados em busca da classe solicitada, na ordem em que foram adicionadas ao depósito. Se um UCL que puder carregar a classe encontrado, ela será retornada; caso contrário, uma *java.lang.ClassNotFoundException* será lançada.

Ainda segundo Fleury, Stark e Norman (2005, p.40) outra fonte de informações útil sobre classes é o próprio *UnifiedLoaderRepository*. Trata-se de um MBean que contém operações para a exibição de informações de classes e pacotes. O depósito padrão está localizado sobre o nome JMX padrão *JMImplementation:name=Default,service=LoaderRepository* e você pode acessar seu MBean através do JMX Console clicando em seu link na página inicial.

Para visualizar as informações de uma determinada classe, você usará a operação *displayClassInfo*, passando o nome totalmente qualificado da classe a ser exibida. Por exemplo, se você usar *org.jboss.jmx.adaptor.html.HTMLAdaptorServlet*, a descrição será exibida. Segundo JBoss (2006) as informações serão uma descarga dos dados da instância de *Class* existentes no depósito e carregadores se ela tiver sido carregada, seguida de carregadores que tiveram o arquivo da classe disponível. Se uma classe for vista com mais de um carregador associado a ela, haverá a possibilidade de erros de carga de classes.



JMX MBean View

MBean Name: **Domain Name:** JMImplementation
service: LoaderRepository
name: Default
 MBean Java Class: org.jboss.mx.loading.UnifiedLoaderRepository3

[Back to Agent View](#) [Refresh MBean View](#)

MBean description:

Management Bean.

List of MBean attributes:

Name	Type	Access	
CacheSize	int	R	2350
URLs	[Ljava.net.URL;	R	[Ljava.net.URL;@1b031
Instance	org.jboss.mx.loading.LoaderRepository	R	org.jboss.mx.loading.Unif
ClassLoadersSize	int	R	35
Translator	org.jboss.util.loading.Translator	RW	

Figura 6 Visualização do MBean *LoaderRepository* da classe padrão no JMX Console

Fonte: JBoss (2006)

Para visualizar as informações de uma determinada classe, você usará a operação *displayClassInfo*, passando o nome totalmente qualificado da classe a ser exibida. Por exemplo, se você usar *org.jboss.jmx.adaptor.html.HtmlAdaptorServlet*, a descrição será exibida. Segundo JBoss (2006) as informações serão uma descarga dos dados da instância de *Class* existentes no depósito e carregadores se ela tiver sido carregada, seguida de carregadores que tiverem o arquivo da classe disponível. Se uma classe for vista com mais de um carregador associado a ela, haverá a possibilidade de erros de carga de classes.

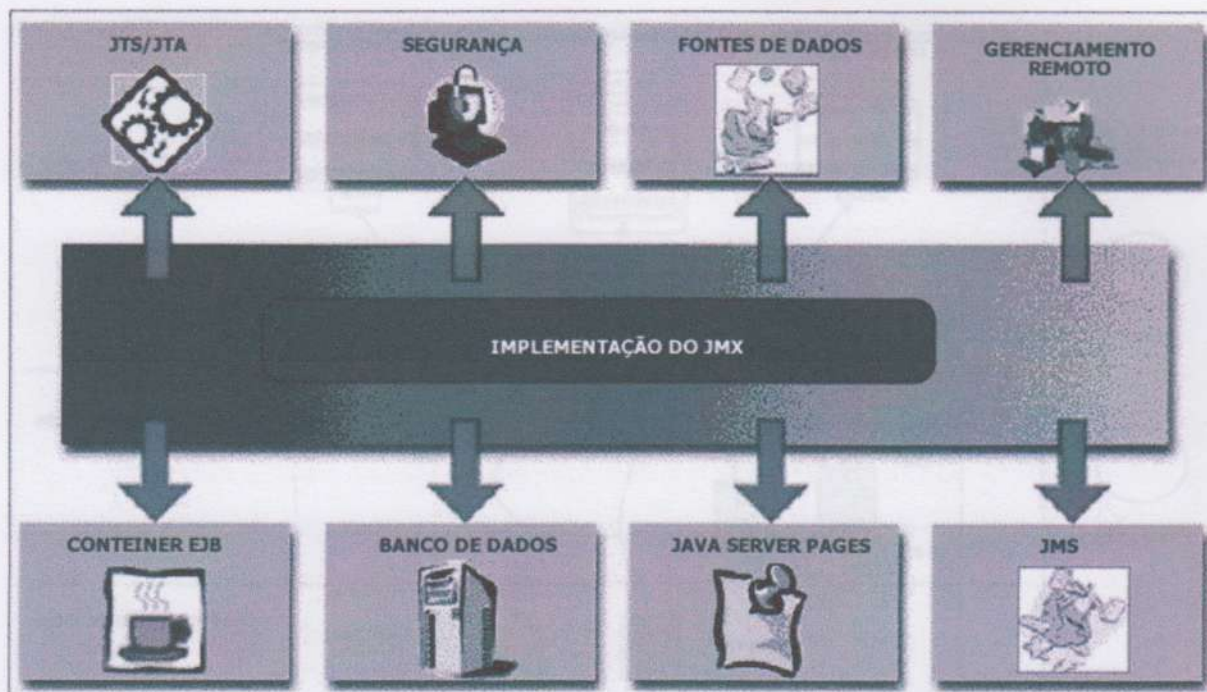


Figura 7 Integração JMX e os componentes padrão do JBoss.
Fonte: JBoss (2006)

O JMX é um padrão para o gerenciamento e monitoração de todo tipo de componentes de hardware e software a partir de Java. Além disso, objetiva fornecer integração com a grande quantidade de padrões de gerenciamento existentes. De acordo com Fleury, Stark e Norman (2005, p.24) existem três níveis no modelo JMX:

- **Instrumentação** — recursos a serem gerenciados.
- **Agentes** — controladores dos objetos de nível de instrumentação.
- **Serviços distribuídos** - mecanismo pelo qual os aplicativos de administração interagem com os agentes e os objetos que eles gerenciam.

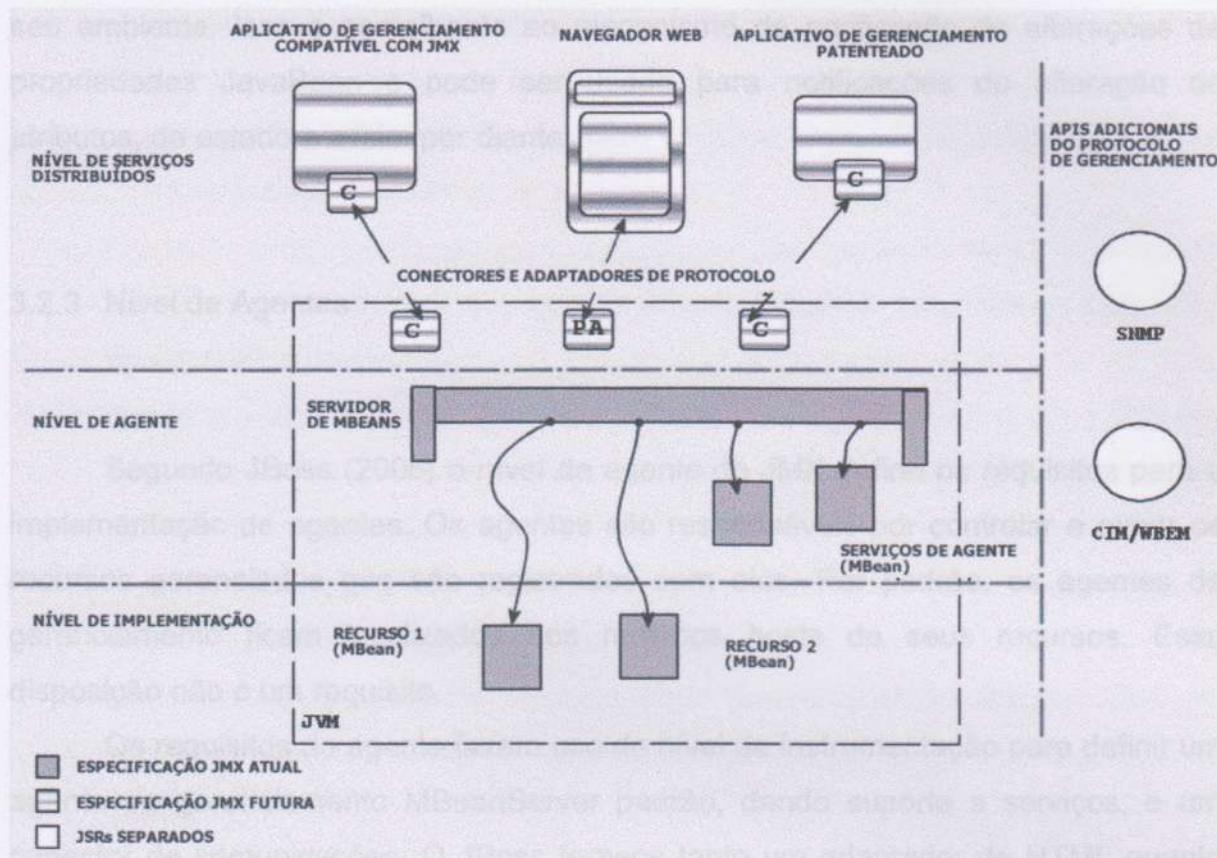


Figura 8 Relacionamentos entre os componentes da arquitetura JMX
Fonte: JBoss (2006)

3.2.2 Nível de Instrumentação

O que vai de acordo com Fleury, Stark e Norman (2005, p.25) o nível de instrumentação define os requisitos para a implementação dos recursos gerenciáveis pelo JMX. Um recurso gerenciável pelo JMX pode ser praticamente qualquer coisa, inclusive aplicativos, componentes de serviço, dispositivos e assim por diante. Ele expõe um objeto ou empacotador Java que descreva suas funções gerenciáveis, o que o torna instrumentado para poder ser gerenciado por aplicativos compatíveis com o JMX.

O usuário fornece a instrumentação de um determinado recurso usando um ou mais beans gerenciados, ou MBeans. Há quatro tipos de implementação de MBeans: padrão, dinâmica, modelo e aberta.

O nível de instrumentação também especifica um mecanismo de notificação. A finalidade desse mecanismo é permitir que os MBeans comuniquem alterações a

seu ambiente. Isso é semelhante ao mecanismo de notificação de alterações de propriedades JavaBean e pode ser usado para notificações de alteração de atributos, de estado e assim por diante.

discutido na especificação JMX atual:

- Fornece uma interface para aplicativos de gerenciamento interagirem

3.2.3 Nível de Agentes

através de um conector.

- Expõe a visualização do gerenciamento de um agente JMX e seus MBeans

Segundo JBoss (2006) o nível de agente do JMX define os requisitos para a implementação de agentes. Os agentes são responsáveis por controlar e expor os recursos gerenciados que são registrados com eles. Por padrão, os agentes de gerenciamento ficam localizados nos mesmos hosts de seus recursos. Essa disposição não é um requisito.

Os requisitos do agente fazem uso do nível de instrumentação para definir um agente de gerenciamento MBeanServer padrão, dando suporte a serviços, e um conector de comunicações. O JBoss fornece tanto um adaptador de HTML quanto de RMI.

Para Fleury, Stark e Norman (2005, p.25) o agente pode ficar situado no hardware hospedeiro dos recursos gerenciáveis pelo JMX quando uma Máquina Virtual Java (JVM, Java Virtual Machine) estiver disponível. É assim que o servidor JBoss usa o MBeanServer. Um agente JMX não precisa saber que recursos ele servirá. Os recursos gerenciáveis pelo JMX poderão usar qualquer agente JMX que ofereça os serviços de que eles precisarem.

Os gerenciadores interagem com os MBeans de um agente através de um adaptador de protocolo ou conector, como descrito na próxima seção. O agente não precisa saber nada sobre os conectores ou aplicativos de gerenciamento que interagem com ele e seus MBeans.

- Componentes do nível de instrumentação

- o. MBeans (padrão, dinâmicos, abstratos e modelo)

3.2.4 Nível de Serviços Distribuídos

- o. Classes de metadados dos MBeans

- Componentes do nível de agente

A especificação JMX informa que uma definição completa do nível de serviços distribuídos está além do escopo de sua versão inicial. A finalidade geral desse nível

e definir as interfaces necessárias a implementação de gerenciadores ou aplicativos de gerenciamento JMX. Segundo Fleury, Stark e Norman (2005, p.25) os seguintes pontos destacam a funcionalidade esperada no nível de serviços distribuídos, como discutido na especificação JMX atual:

- Fornece uma interface para aplicativos de gerenciamento interagirem transparentemente com um agente e seus recursos gerenciáveis pelo JMX através de um conector.
- Expõe a visualização do gerenciamento de um agente JMX e seus MBeans convertendo o significado de sua semântica nas estruturas de um protocolo rico em dados (por exemplo, HTML, SNMP).
- Distribui informações de gerenciamento de plataformas de gerenciamento de alto nível para vários agentes JMX.
- Consolida informações de gerenciamento vindas de vários agentes JMX em exibições lógicas que sejam relevantes as operações empresariais do usuário final.
- Fornece segurança.

Os componentes do nível de serviços distribuídos foram projetados para permitir o gerenciamento cooperativo de redes de agentes e seus recursos. Esses componentes podem ser expandidos para fornecer um aplicativo de gerenciamento completo.

3.2.5 Componentes do JMX

De acordo com JBoss (2006) os componentes JMX podem ser agrupados por nível:

- Componentes do nível de instrumentação
 - MBeans (padrão, dinâmicos, abertos e modelo)
 - Elementos do modelo de notificações
 - Classes de metadados dos MBeans
- Componentes do nível de agente
 - O servidor de MBeans

- Serviços de agentes

3.2.5.1 MBeans

Segundo JBoss (2006) um MBean é um objeto Java que implementa uma das interfaces MBean padrão e segue os padrões de projeto associados. O MBean de um recurso expõe todas as informações e operações necessárias de que um aplicativo de gerenciamento precisa para controlar o recurso.

O escopo da interface de gerenciamento de um MBean inclui:

- Valores de atributos que podem ser acessados por nome
- Operações ou funções que podem ser chamadas
- Notificações ou eventos que podem ser emitidos
- Os construtores da classe Java de MBeans

De acordo com JBoss (2006) o JMX define quatro tipos de MBeans para dar suporte a diferentes necessidades de instrumentação:

- **MBeans padrão** - Usam uma convenção de nomeação simples de estilo Java e uma interface definida estaticamente é o tipo mais comum de MBean usado pelo JBoss.
- **MBeans dinâmicos** - Devem implementar a interface *javax.management.DynamicMBean* e expõem sua interface de gerenciamento no tempo de execução, quando o componente é instanciado, para maior flexibilidade. O JBoss faz uso MBeans dinâmicos nos casos em que os componentes a serem gerenciados não são conhecidos até o tempo de execução.
- **MBeans abertos** - São uma extensão dos MBeans dinâmicos. Os MBeans abertos se baseiam em tipos de dados básicos, autodescritivos e amigáveis para fornecer gerenciamento universal.
- **MBeans modelo** — Também são uma extensão dos MBeans dinâmicos. Os MBeans modelo devem implementar a interface *javax.management.modelmbean.ModelMBean*. Eles simplificam a

instrumentação de recursos fornecendo comportamento padrão. Os XMBeans do JBoss são uma implementação de MBeans modelo.

3.2.5.4 O servidor de MBeans

3.2.5.2 Elementos do modelo de notificações

Conforme Fleury, Stark e Norman (2005, p.27) um componente-chave do nível de agente é o servidor de MBeans. Sua funcionalidade é exposta através de uma

JBoss (2006) define as notificações JMX como uma extensão do modelo de eventos Java. Tanto o servidor de MBeans quanto os MBeans podem enviar notificações para fornecer informações. A especificação JMX define o objeto de evento *Notification* do pacote *javax.management*, o emissor de eventos *NotificationBroadcaster* e as interfaces receptoras de eventos *NotificationListener*. A especificação também define as operações do servidor de MBeans que permitem o registro de ouvintes de notificações.

Para JBoss (2006) os MBeans podem ser instanciados e registrados no servidor de MBeans pelos seguintes elementos:

3.2.5.3 Classes de metadados do MBean

- O próprio agente
- Um aplicativo de gerenciamento remoto (através dos serviços

Para Fleury, Stark e Norman (2005, p.27) define como um conjunto de classes de metadados que descrevem a interface de gerenciamento de um MBean. Os usuários podem obter uma exibição em comum dos metadados de qualquer um dos quatro tipos de MBeans consultando o servidor com o qual os MBeans foram registrados. As classes de metadados abrangem os atributos, operações, notificações e construtores de um MBean. Para cada um desses itens, os metadados incluem um nome, uma descrição e suas características específicas. Por exemplo, uma característica de um atributo seria se ele pode ser lido, gravado ou ambos. Os metadados de uma operação contém a assinatura de seu parâmetro e tipos de retorno.

Os diferentes tipos de MBeans estendem as classes de metadados para poder fornecer informações adicionais, quando necessário. Essa herança comum torna as informações padrão disponíveis, independentemente do tipo de MBean. Um aplicativo de gerenciamento que souber como acessar as informações estendidas de um tipo específico de MBean poderá fazer isso.

3.2.5.5 Serviços de agente

3.2.5.4 O servidor de MBeans

De acordo com Fleury, Stark e Norman (2005, p.28) os serviços de agente do JMX. Conforme Fleury, Stark e Norman (2005, p.27) um componente-chave do nível de agente é o servidor de MBeans. Sua funcionalidade é exposta através de uma instância de *javax.management.MBeanServer*. O servidor de MBeans é um registro de MBeans que torna a interface de gerenciamento de MBeans disponível para ser usada por aplicativos de gerenciamento. O servidor de MBeans nunca expõe diretamente o próprio objeto MBean; em vez disso, uma interface de gerenciamento de MBeans é exposta através de metadados e operações disponíveis na interface do servidor de MBeans. Isso fornece uma ligação livre entre os aplicativos de gerenciamento e os MBeans que eles gerenciam.

Para JBoss (2006) os MBeans podem ser instanciados e registrados no servidor de MBeans pelos seguintes elementos:

- Outro MBean
- O próprio agente
- Um aplicativo de gerenciamento remoto (através dos serviços distribuídos)

Quando você registrar um MBean, terá que atribuir a ele um nome de objeto exclusivo. Esse nome se tornará o identificador exclusivo pelo qual aplicativos de gerenciamento detectarão o objeto em que executarão operações de gerenciamento.

De acordo com JBoss (2006) as operações disponíveis em MBeans através do servidor de MBeans incluem as seguintes:

- Descoberta das interfaces de gerenciamento de MBeans
- Leitura e gravação de valores de atributos
- Chamada de operações definidas por MBeans
- Registro de eventos de notificação
- Consulta de MBeans baseada em seus nomes de objeto e valores de atributos

3.2.5.5 Serviços de agente

De acordo com Fleury, Stark e Norman (2005, p.28) os serviços de agente do JMX são objetos que dão suporte a operações padrão nos MBeans registrados no servidor de MBeans. Inclusões de suporte a serviços de gerenciamento o ajudarão a construir soluções de gerenciamento mais poderosas. Geralmente próprios serviços de agente são MBeans, o que permite que o agente e sua funcionalidade sejam controlados através do servidor de MBeans. JBoss (2006) diz que a especificação JMX define os seguintes serviços de agente:

- **Um serviço dinâmico de carga de classes MLet (management applet) -** Esse serviço permite a recuperação e instanciação de novas classes e bibliotecas nativas a partir de um local arbitrário na rede.
- **Serviços de monitor** - Esses serviços observam o valor de string ou numérico do atributo de um MBean e podem notificar outros objetos sobre vários tipos de alterações no alvo.
- **Serviços de timer** — Esses serviços fornecem um mecanismo de agendamento baseado em uma notificação de alarme único ou em uma notificação periódica, repetida.
- **O serviço de relacionamento** - Esse serviço define associações entre MBeans e garante consistência nos relacionamentos.

3.3 JBOSS JNDI

O serviço de nomes do JBoss desempenha um papel-chave no J2EE, porque fornece um recurso que permite que o usuário converta um nome em um objeto. Sendo essa uma necessidade fundamental em qualquer ambiente de programação porque os programadores e administradores querem ter a possibilidade de referenciar objetos e serviços por nomes reconhecíveis. Um bom exemplo de um serviço de nomes conhecido é o DNS (Domain Name System) da internet. O DNS permite a referência a hosts com o uso de nomes lógicos em vez de endereços

numéricos na internet. O JNDI desempenha um papel semelhante no J2EE permitindo que os desenvolvedores e administradores criem vinculações de nome e objeto a serem usadas em componentes J2EE.

3.3.1 Tratamento de Nomes no JNDI

A noção de nome é fundamental no JNDI. O sistema de nomes determina a sintaxe que o nome deve seguir. A sintaxe do sistema de nomes permite que o usuário converta representações em string de nomes em seus componentes. Um nome é usado com um sistema de nomes na localização de objetos, que poderia ser descrito como um conjunto de objetos que tem nomes exclusivos, sendo que para localizar um objeto num sistema de nomes, basta fornecerem um nome ao sistema para que ele retorne o depósito de objetos associado ao nome.

3.3.2 Contextos

Segundo Fleury, Stark e Norman (2005, p.110) a interface *javax.naming.Context* é a principal interface para interação com um serviço de nomes. A interface *Context* representa um conjunto de vinculações de nome e objeto. Todo contexto tem uma convenção de nomes associada que determina como ele divide nomes de string em instâncias de *javax.naming.Name*. Para criar uma vinculação de nome e objeto, chame o método de vinculação de um contexto e especifique um nome e um objeto como argumentos. Sendo possível recuperar o objeto usando seu nome, através do método de pesquisa de *Context*.

Ele fornece uma espinha dorsal comum que permitira que você integre módulos, contêineres e plug-ins. Os componentes são declarados como serviços MBean e, em seguida, carregados no JBoss. Subsequentemente podem ser administrados com o uso do JMX.

• *NamingService* — fornece o serviço de nomenclatura JNDI. Esse é um serviço chave que é usado universalmente pelos componentes da tecnologia J2EE.

3.3.3 Arquitetura JBossNS

JBoss (2006) diz que a arquitetura JBossNS é uma implementação Java baseada em soquete/RMI da interface *javax.naming.Context*. É uma implementação cliente/servidor que pode ser acessada remotamente. A implementação é otimizada para que o acesso de dentro da mesma VM em que o servidor JBossNS estiver sendo executado não envolva soquetes. O acesso na mesma VM ocorre através de uma referência de objeto que está disponível como um singleton global.

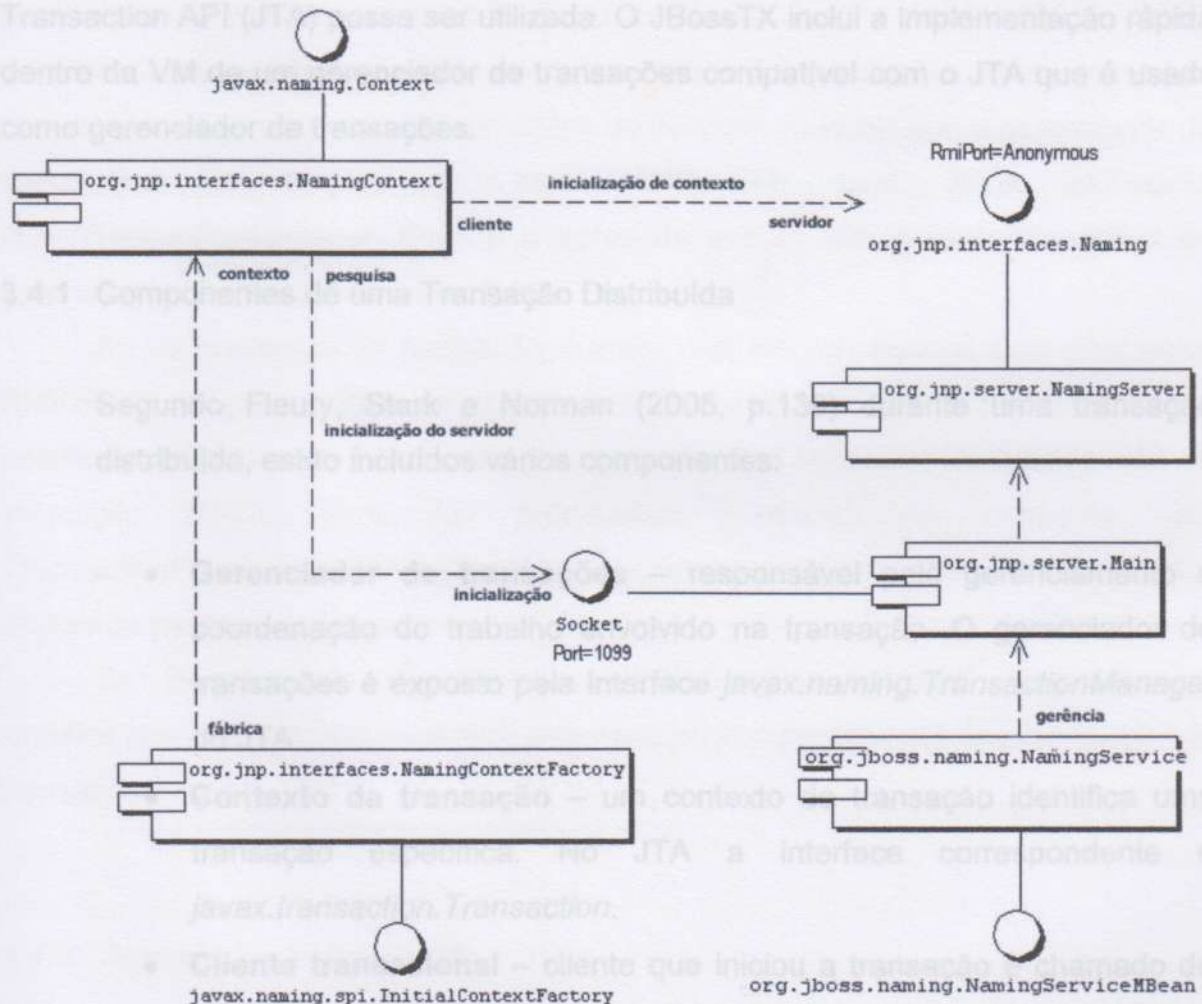


Figura 9 Principais componentes da arquitetura JBossNS.

Fonte: JBoss (2006)

Componentes chave da arquitetura JBossNS:

- **NamingService** — fornece o serviço de nomeação JNDI. Esse é um serviço-chave que é usado universalmente pelos componentes da tecnologia J2EE.

- **NamingContextFactory** — usada para criar a instância de *Context* que será empregada em operações subseqüentes, sendo ela a implementação do JBossNS para a interface *javax.naming.spi.InitialContextfactory*.

3.4 JBOSS TRANSAÇÕES

3.4.2 Detalhes internos das transações no JBoss

De acordo com Fleury, Stark e Norman (2005, p.137) a arquitetura JBossTX permite que qualquer implementação de gerenciador de transações do Java Transaction API (JTA) possa ser utilizada. O JBossTX inclui a implementação rápida dentro da VM de um gerenciador de transações compatível com o JTA que é usado como gerenciador de transações.

3.4.1 Componentes de uma Transação Distribuída

Segundo Fleury, Stark e Norman (2005, p.138) durante uma transação distribuída, estão incluídos vários componentes:

- **Gerenciador de transações** – responsável pelo gerenciamento e coordenação do trabalho envolvido na transação. O gerenciador de transações é exposto pela interface *javax.naming.TransactionManager* do JTA.
- **Contexto da transação** – um contexto de transação identifica uma transação específica. No JTA a interface correspondente é *javax.transaction.Transaction*.
- **Cliente transacional** – cliente que iniciou a transação é chamado de *originador de transação*. O cliente podendo ser um usuário explícito ou implícito de interfaces JTA sem que haja interface que o represente.
- **Objeto transacional** – objeto cujo comportamento é afetado por operações executadas nele dentro de um contexto transacional. O objeto também pode ser um cliente transacional.

- **Recurso recuperável** – é um objeto transacional cujo estado será salvo em armazenamento se a transação for confirmada, podendo ser redefinido para o estado apresentado no início da transação se esta for revertida.

3.4.2 Detalhes internos das transações no JBoss

Como o servidor de aplicações JBoss foi criado para ser independente do gerenciador de transações utilizado. Segundo Fleury, Stark e Norman (2005, p.140) o JBoss usa a interface JTA *javax.transaction.TransactionManager* como sua concepção do gerenciador de transações do servidor. Sempre que o gerenciador de transações for usado, ele será obtido no local JNDI conhecido, *java:/TransactionManager*. Esse é o ponto de acesso globalmente disponível do gerenciador de transações no servidor.

Se os contextos de transação tiverem que ser propagados com chamadas RMI/JRMP, o gerenciador de transações também terá que implementar duas interfaces simples para importação e exportação dos *contextos de propagação de transação* (TPCs, *transaction propagation contexts*). As interfaces são: *TransactionPropagationContextImporter* e *TransactionPropagationContextFactory*, ambas do pacote *org.jboss.tm*.

Ser independente do gerenciador de transações realmente usado também significa que o JBoss não especifica o formato ou tipo de contexto de propagação de transações usado.

3.4.3 Gerenciador de transações padrão

O JBoss é por padrão configurado para usar o gerenciador de transações de alta performance da VM. Segundo Fleury, Stark e Norman (2005, p.141) o gerenciador é muito rápido, mas tem duas limitações:

- Não executa registro transacional e, portanto, é incapaz de se recuperar de maneira automatizada após interrupção no servidor.
- Oferece suporte a contextos de propagação de transação com chamadas remotas, porém não dá suporte a esses contextos de outras máquinas virtuais, portanto, todo trabalho transacional deve ser executado na mesma máquina virtual do servidor JBoss.

3.5.1 Modelo de segurança do JBoss

O serviço MBean corresponde ao gerenciador de transações padrão é o MBean `org.jboss.tm.TransactionManagerService`. Ele Possui quatro atributos configuráveis:

- **TransactionTimeout** – tempo-limite padrão da transação em segundos. Sendo o valor padrão de 300 segundos (cinco minutos)
- **InterruptThreads** – indica se o gerenciador de transações deve interromper segmentos quando a transação expirar. Sendo o valor padrão *false*.
- **GlobalIdsEnabled** – indica se o gerenciador de transações deve usar identificações de transação global. Deve ser configurado como verdadeiro para a definição de transações de IIOP. Sendo o valor padrão verdadeiro.
- **XidFactory** – o `ObjectName` JMX do serviço MBean que fornecerá a implementação de `org.jboss.tm.XidFactoryMBean`. a interface `XidFactoryMBean` é usada para criar instâncias de `javax.transaction.xa.Xid`. É um paliativo para drivers XA do JDBC que só funcionam com suas próprias implementações Xid.

3.5 JBOSS SEGURANÇA

De acordo com JBoss (2006) o modelo de segurança que a especificação J2EE sugere é um modelo declarativo. Ele é declarativo pelo fato de ser preciso descrever as funções e permissões de segurança com o uso de um descritor XML

Fonte: JBOSS INC (2006)

padrão em vez de a segurança ser embutida no componente operacional. Isso isola a segurança do código de nível operacional porque ela tenda a depender mais de onde o componente é implantado do que a ser um aspecto inerente à lógica operacional do componente.

3.5.1 Modelo de segurança do JBoss

A segurança de nível mais baixo é definida como um conjunto de interfaces para as quais as implementações alternativas podem ser fornecidas. Segundo JBoss (2006) três interfaces básicas definem a camada de segurança do servidor JBoss: *org.jboss.security.AuthenticationManager*, *org.jboss.security.RealMapping* e *org.jboss.security.SecurityProxy*.

3.5.2 Configurando o JBoss para ser usado atrás de firewall

O JBoss vem com muitos serviços baseados em soquete que abrem portas de escuta. Abaixo segue uma lista dos serviços que abrem porta cujo funcionamento você pode ter que configurar quando acessar o JBoss através de um firewall.

Porta	Tipo	Serviço
1099	TCP	<i>org.jboss.naming.NamingService</i>
1098	TCP	<i>org.jboss.naming.NamingService</i>
4444	TCP	<i>org.jboss.invocation.jmp.server.JRMPInvoker</i>
4445	TCP	<i>org.jboss.invocation.pooled.server.PooledInvoker</i>
8009	TCP	<i>org.jboss.web.tomcat.tc4.EmbeddedTomcatService</i>
8080	TCP	<i>org.jboss.web.tomcat.tc4.EmbeddedTomcatService</i>
8083	TCP	<i>org.jboss.web.WebService</i>
8093	TCP	<i>org.jboss.mq.il.uil2.UILServerILService</i>

Quadro 3 Portas de configuração padrão
Fonte: JBOSS INC (2006)

Porta	Tipo	Serviço
1100	TCP	<i>org.jboss.ha.jndi.HaNamingService</i>
0	TCP	<i>org.jboss.ha.jndi.HaNamingService</i>
1102	UDP	<i>org.jboss.ha.jndi.HaNamingService</i>
1161	UDP	<i>org.jboss.jmx.adaptor.snmp.agent.SnmpAgentService</i>
1162	UDP	<i>org.jboss.jmx.adaptor.snmp.trapd.TrapdService</i>
3528	TCP	<i>org.jboss.invocation.iiop.IIOPInvoker</i>
45566	UDP	<i>org.jboss.ha.framework.server.ClusterPartition</i>

Quadro 4 Portas adicionais na configuração de servidor tipo *all*

Fonte: JBOSS INC (2006)

3.5.3 Protegendo o JBoss

O JBoss vem com vários pontos de acesso admin que tem que ser protegidos ou removidos para impedir o acesso a funções admin em uma implantação.

3.5.3.1 Serviço *jmx-console.war*

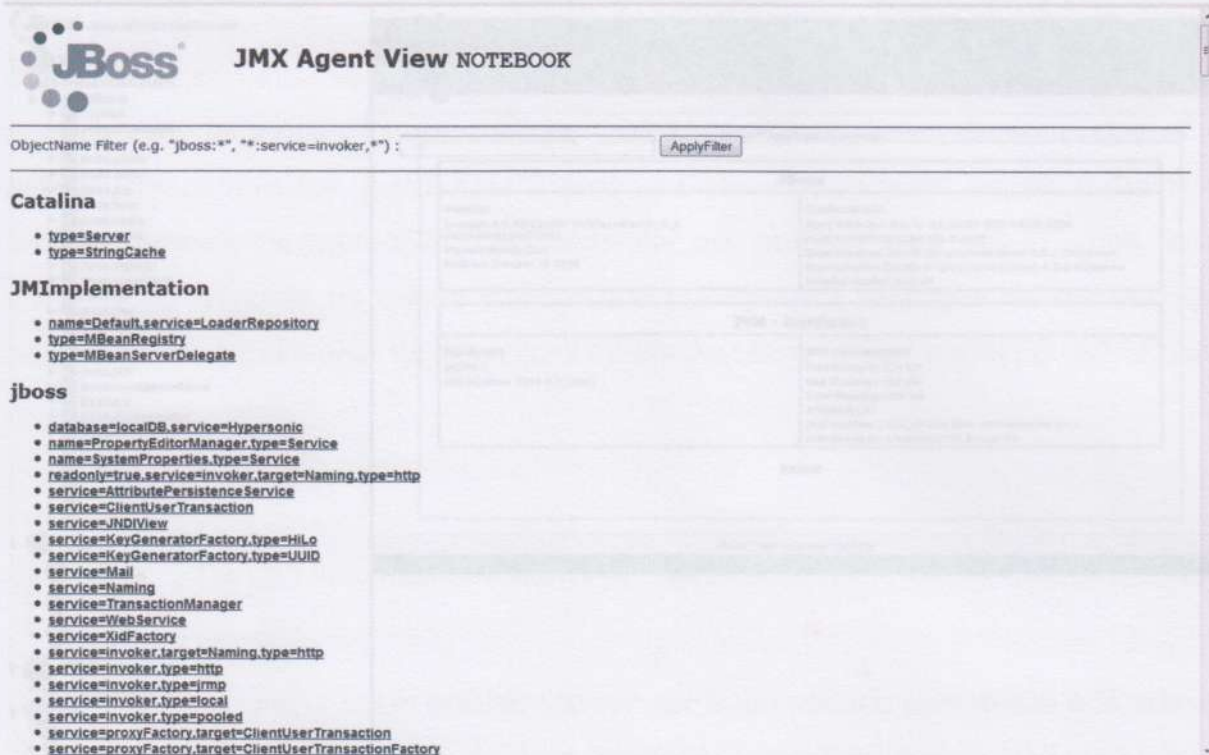


Figura 10 Visualização da aplicação JMX-CONSOLE
Fonte: Dados da pesquisa

O arquivo *jmx-console.war* encontrado no diretório de implantação fornece uma visualização em HTML do microkernel JMX. Portanto, disponibiliza acesso arbitrário de tipo admin, como o usado na interrupção do servidor, interrupção de serviços, implantação de novos serviços e assim por diante. Ele deve ser protegido como qualquer outro aplicativo Web ou removido.

3.5.3.2 Serviço *web-console.war*

3.5.3.3 Serviço *http-invoker.sar*

O arquivo *http-invoker.sar* encontrado no diretório de implantação é um serviço que fornece acesso RMI/HTTP a EJBs e ao serviço de nomes JNDI. Ele inclui um serviço que processa envios de objetos *org.jboss.invocation.Invocation* reconstituídos representando chamadas que devem ser despatchadas para *MBeanServer*. Isso efetivamente concede acesso a MBeans que dão suporte a

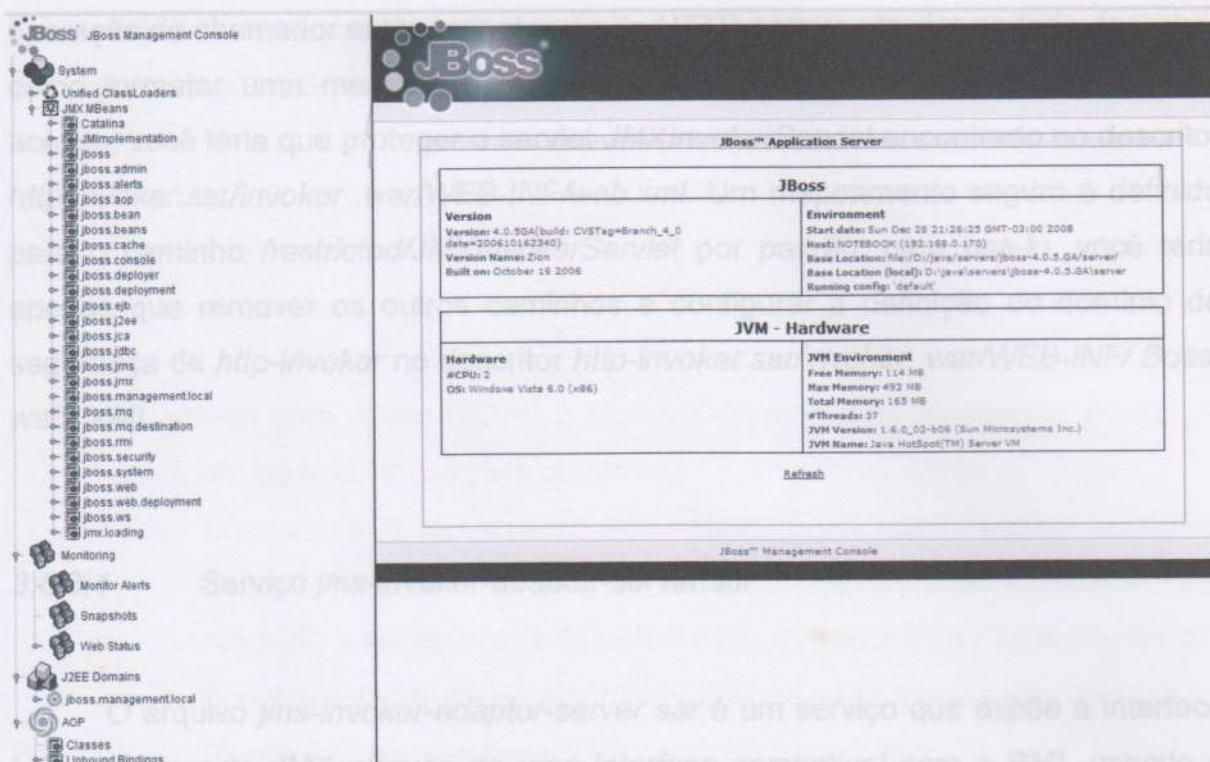


Figura 11 Visualização da aplicação WEB-CONSOLE
Fonte: Dados da pesquisa

O arquivo *web-console.war* encontrado no diretório *deploy/management* é outra visualização de aplicativo Web microkernel JMX. Ele usa uma combinação de applet e visualização HTML e fornece o mesmo nível de acesso a funcionalidade admin que *jmx-console.war*. Portanto, deve ser protegido ou removido. O arquivo *web-console.war* contém modelos desativados para a segurança básica de seu descritor *WEB-INF/web.xml*, assim como uma configuração desativada para domínio de segurança de *WEB-INF/jboss-web.xml*.

3.5.3.3 Serviço *http-invoker.sar*

O arquivo *http-invoker.sar* encontrado no diretório de implantação é um serviço que fornece acesso RMI/HTTP a EJBs e ao serviço de nomes JNDI. Ele inclui um servlet que processa envios de objetos *org.jboss.invocation.Invocation* reconstituídos representando chamadas que devem ser despachadas para MBeanServer. Isso efetivamente concede acesso a MBeans que dão suporte a

operação de chamador autônomo através do HTTP porque alguém poderia descobrir como formatar uma mensagem HTTP apropriada. Para proteger esse ponto de acesso, você teria que proteger o servlet *JMXInvokerServlet* encontrado no descritor *http-invoker.sar/invoker.war/WEB-INF/web.xml*. Um mapeamento seguro é definido para o caminho */restricted/JMXInvokerServlet* por padrão; para usá-lo, você teria apenas que remover os outros caminhos e configurar a definição do domínio de segurança de *http-invoker* no descritor *http-invoker.sar/invoker.war/WEB-INF/Boss-web.XML*.

3.5.3.4 Serviço *jms-invoker-adaptor-server.sar*

O arquivo *jms-invoker-adaptor-server.sar* é um serviço que expõe a interface *MBeanServer* do JMX através de uma interface compatível com o RMI, usando o serviço de chamador autônomo RMI/JRMP. Atualmente, a única maneira de tal serviço ser protegido seria alterar o protocolo para o RMI/HTTP e proteger *http-invoker.sar*. No futuro, esse serviço será implantado como um *XMBean* com um interceptador de segurança que dará suporte a verificações de acesso baseadas em funções.

3.6 JBOSS CLUSTERING

Conforme JBoss (2006) um cluster é um conjunto de nós. Em um cluster JBoss, um nó é uma instância de servidor JBoss. Assim, para construir um cluster, as instâncias do JBoss devem ser agrupadas, este agrupamento é conhecido como uma partição. Em uma mesma rede, podem ter diferentes clusters. De modo que para diferenciá-los, cada cluster deve ter um nome único.

Clustering permite executar várias aplicações em servidores de forma paralela (também conhecido por cluster de nós). A carga está distribuída em diferentes servidores, e caso ocorre de qualquer um dos servidores falhar, a aplicação ainda é acessível através de outros nós do cluster. Clustering é crucial para aplicações que

deve ser altamente escaláveis, desta maneira você pode melhorar o desempenho da aplicação, simplesmente adicionando mais nós ao cluster.

3.6.1 Características



De acordo com JBoss (2006) o servidor de aplicação atualmente possui as seguintes características para clustering:

- Descoberta Automática de Nós - Nós em um cluster podem encontrar uns aos outros sem necessidade de configuração adicional, não havendo a necessidade de definir estaticamente a topologia do cluster.
- Funcionalidades para Tolerância a Falhas e Balanceamento de Carga
- Replicação de sessões HTTP com Tomcat 4.1 and 5.0
- Clientes JNDI podem automaticamente descobrir o JNDI InitialContext.
- Suporte a stateless session beans
- Suporte a stateful sessions beans
- Suporte a entity beans

3.6.2 Partição

Como descrito anteriormente, um cluster é um conjunto de nós. Desta maneira, para construir um cluster, as várias instâncias do JBoss têm de ser agrupadas em uma partição.

JBoss (2006) diz que a partição é o conceito central para a definição de cluster no JBoss, em uma mesma rede, podem existir diferentes partições. De modo que para diferenciá-las, cada partição deve ter um nome único.

- Detectar novos nós ou nós que falharam ou nós desativados

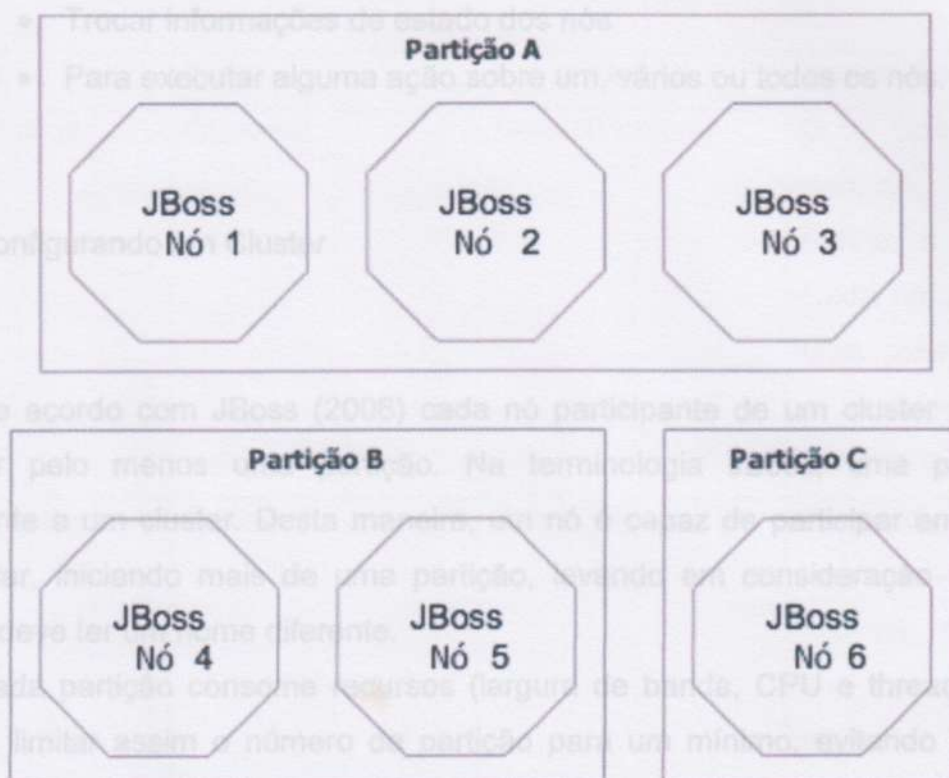


Figura 12 Exemplo de instâncias de servidores JBoss divididas em partições

Fonte: JBoss (2006)

Na figura acima, um dos clusters mostra um caso interessante a ser analisado: ele é composto de um único nó, apesar de tolerância a falhas ou balanceamento de carga não serem possíveis, um novo nó poderia ser acrescentado a qualquer momento para esta partição, o que a partir de então poderia fornecer aqueles serviços.

Também é possível para uma instância do JBoss fazer parte de várias partições ao mesmo tempo.

Se nenhum nome for atribuído a uma partição, ela utilizará um nome padrão. Conseqüentemente, à medida que forem sendo adicionados novos nós sem que seja especificado um nome para a partição, eles serão inicializados como pertencentes a partição padrão do cluster.

A descoberta automática de nós em um cluster e a comunicação são altamente configuráveis e podem, portanto, serem aplicados a diferentes topologias de redes (LAN, WAN, etc). Os nós em um cluster precisam se comunicar por diversas razões:

- Detectar novos nós ou nós que falharam ou nós desativados

- Trocar informações de estado dos nós
- Para executar alguma ação sobre um, vários ou todos os nós.

3.6.3 Configurando um Cluster

De acordo com JBoss (2006) cada nó participante de um cluster no JBoss deve ter pelo menos uma partição. Na terminologia JBoss, uma partição é equivalente a um cluster. Desta maneira, um nó é capaz de participar em mais de um cluster, iniciando mais de uma partição, levando em consideração que cada partição deve ter um nome diferente.

Cada partição consome recursos (largura de banda, CPU e threads), desta maneira, limitar assim o número de partição para um mínimo, evitando a criação desnecessária de partições é uma boa recomendação levando em conta a economia de recursos.

O JBoss vem com três diferentes configurações servidor: *minimal*, *default* e de *all*. Porém, o Clustering só está habilitado na configuração *all*. Na pasta */deploy* existe um arquivo chamado *cluster-service.xml* que descreve a configuração para a partição padrão do cluster.

```
<mbean code="org.jboss.ha.framework.server.ClusterPartition"
name="jboss:service=${jboss.partition.name:DefaultPartition}">
```

Quadro 5 Definição do Clustering MBean

Fonte: Dados da pesquisa

```
<code>
<UDP mcast_addr="${jboss.partition.udpGroup:228.1.2.3}"
mcast_port="${jboss.partition.mcast_port:45566}"
ttl="8"
ucast_rcv_buf_size="2000000"
ucast_snd_buf_size="64000"
mcast_rcv_buf_size="25000000"
mcast_snd_buf_size="64000"
loopback="false"
discard_incompatible_packets="true"
enable_huntling="false"
max_bundle_size="64000"
max_bundle_timeout="30"
use_incoming_packet_handler="true"
</code>
```

Atributo	Tipo	Valor padrão	Descrição
PartitionName	Opcional	DefaultPartition	O nome da partição (a qual define o cluster). Cada nó que inicia uma partição com o mesmo nome, compõe o cluster.
DeadlockDetection	Opcional	False	Propriedade boolean que informa se deve ser executado um algoritmo de detecção de deadlock a cada solicitação
PartitionConfig	Opcional	Ver quadro abaixo	Propriedade string que define o conjunto de protocolos usados na configuração

Quadro 6 Parâmetros de configuração do arquivo cluster-service.xml
 Fonte: JBoss (2006)

3.5.4 Farming Deployment

<Config>

```

<UDP mcast_addr="$jboss.partition.udpGroup:228.1.2.3"
      mcast_port="$jboss.hapartition.mcast_port:45566"
      tos="8"
      ucast_rcv_buf_size="20000000"
      ucast_send_buf_size="640000"
      mcast_rcv_buf_size="25000000"
      mcast_send_buf_size="640000"
      loopback="false"
      discard_incompatible_packets="true"
      enable_bundling="false"
      max_bundle_size="64000"
      max_bundle_timeout="30"
      use_incoming_packet_handler="true"
  </UDP>
  </Config>
  
```

```

        use_outgoing_packet_handler="false"
        ip_ttl="$jgroups.udp.ip_ttl:2}"
        down_thread="false" up_thread="false"/>
    <PING timeout="2000"
        down_thread="false" up_thread="false" num_initial_members="3"/>
    <MERGE2 max_interval="100000"
        down_thread="false" up_thread="false" min_interval="20000"/>
    <FD_SOCK down_thread="false" up_thread="false"/>
    <FD timeout="10000" max_tries="5" down_thread="false" up_thread="false"
shun="true"/>
    <VERIFY_SUSPECT          timeout="1500"          down_thread="false"
up_thread="false"/>
    <pbcast.NAKACK max_xmit_size="60000"
        use_mcast_xmit="false" gc_lag="0"
        retransmit_timeout="300,600,1200,2400,4800"
        down_thread="false" up_thread="false"
        discard_delivered_msgs="true"/>
    <UNICAST timeout="300,600,1200,2400,3600"
        down_thread="false" up_thread="false"/>
    <pbcast.STABLE stability_delay="1000" desired_avg_gossip="50000"
        down_thread="false" up_thread="false"
        max_bytes="400000"/>
    <pbcast.GMS print_local_addr="true" join_timeout="3000"
        down_thread="false" up_thread="false"
        join_retry_timeout="2000" shun="true"
        view_bundling="true"/>
    <FRAG2 frag_size="60000" down_thread="false" up_thread="false"/>
    <pbcast.STATE_TRANSFER    down_thread="false"    up_thread="false"
use_flush="false"/>
</Config>

```

Quadro 7 Conjunto de protocolos utilizados na configuração da partição
 Fonte: JBoss (2006)

3.6.4 Farming Deployment

Com o JBoss Clustering habilitado você pode fazer o uso de hot-deployment em farm, o que significa que ao fazer a publicação de uma aplicação através de um EAR, WAR ou JAR no diretório *deploy* em um dos nós do cluster, a aplicação será automaticamente publicada em todos os nós do cluster.

Farm é ativado por padrão configuração de servidor *all*, fazendo com que você não precise configurá-lo manualmente. Também é possível criar o arquivo XML e configurá-lo manualmente conforme quadro abaixo, lembrando de copiá-lo para o

diretório `$JBoss_HOME/server/all/deploy/deploy.last/` com o nome `farm-service.xml`.

```
<?xml version="1.0" encoding="UTF-8"?>
<server>
<classpath codebase="lib" archives="jbossa.jar"/>
<mbean code="org.jboss.ha.framework.server.FarmMemberService"
name="jboss:service=FarmMember,partition=DefaultPartition">
...
<attribute name="PartitionName">DefaultPartition</attribute>
<attribute name="ScanPeriod">5000</attribute>
<attribute name="URLs">farm/</attribute>
</mbean>
</server>
```

Quadro 8 Conteúdo do arquivo `farm-service.xml`

Fonte: JBoss (2006)

Atributo	Tipo	Valor padrão	Descrição
PartitionName	Obrigatório	DefaultPartition	O nome da partição a qual ele pertence.
URLs	Obrigatório	farm/	Diretório onde os verificadores de arquivos serão publicados. O diretório será criado caso ainda não exista.
ScanPeriod	Obrigatório	5000	Intervalo que o diretório deve ser verificado por mudanças

Quadro 9 Parâmetros do arquivo de configuração `farm-service.xml`

Fonte: JBoss (2006)

CONCLUSÃO

REFERÊNCIAS

Entre as principais conclusões obtidas com este trabalho, está a constatação de que tem ocorrido uma rápida evolução das tecnologias empregadas no desenvolvimento de sistemas corporativos nos últimos anos e de que as empresas têm encontrado dificuldades em se manterem atualizadas em relação às novas tecnologias, devido à escassez de recursos disponíveis. Em uma conjuntura onde se tem cada vez orçamentos menores, mas em contrapartida têm-se pressões por resultados cada vez maiores, a utilização de um servidor de aplicações gratuito de alta qualidade, o JBoss, assim como o Linux e Apache, também exemplos de softwares de código aberto, representam uma excelente alternativa coerente com o cenário de TI das grandes corporações.

Com relação a trabalhos futuros que poderiam ser desenvolvidos dentro do mesmo escopo do presente trabalho, pode-se sugerir a comparação entre diferentes servidores de aplicação de código aberto em diferentes sistemas operacionais, para que se pudesse concluir, de forma mais precisa, quais são as principais vantagens e desvantagens de cada servidor, principalmente com relação à performance. Um estudo deste tipo seria de grande utilidade para as empresas e ou usuários que hoje têm dificuldade na escolha de um servidor de aplicação que atenda suas necessidades.

Acesso: 18 novembro 2008

JavaServer Faces Technology, 2006. Disponível em

<<http://java.sun.com/javasee/javaserverfaces/>>

Acesso: 28 novembro 2008

Simplified Guide to the Java™ 2 Platform, Enterprise Edition, 1999.

Disponível em <http://java.sun.com/j2ee/reference/whitepapers/j2ee_guide.pdf>

Acesso: 13 novembro 2008

The Java™ EE 5 Tutorial, 2006. Disponível em

<<http://java.sun.com/javasee/5/docs/tutorial/doc/>>

Acesso: 15 novembro 2008.

REFERÊNCIAS

FLEURY, Marc; STARK, Scott; NORMAN, Richards; JBoss® 4.0 Guia Oficial; Alta Books, 2005.

GOSLING, J.; MCGILTON, H. The Java Language Environment: Contents, 1996. Disponível em <<http://java.sun.com/docs/white/langenv/index.html>>. Acesso: 12 novembro 2008

JBoss INC. JBoss Run Parameters, 2004. Disponível em <<http://www.jboss.org/community/docs/DOC-10749>>. Acesso: 28 dezembro 2008

_____. The JBoss 4 Application Server Guide Release 5, 2006. Disponível em <<http://docs.jboss.org/jbossas/jboss4guide/r5/html/index.html/>>. Acesso: 17 novembro 2008

STEARNS, J.; CHINNICI, R.; SAHOO. Update: An Introduction to the Java EE 5 Platform, 2006. Disponível em <http://java.sun.com/developer/technicalArticles/J2EE/intro_ee5/>. Acesso: 20 novembro 2008

SUN MICROSYSTEMS. Questions and Answers – Enterprise Information System Tier. Disponível em <http://java.sun.com/blueprints/qanda/eis_tier/index.html>. Acesso : 19 novembro 2008

_____. JavaServer Faces Technology, 2006. Disponível em <<http://java.sun.com/javaee/javaserverfaces/>>. Acesso: 28 novembro 2008

_____. Simplified Guide to the Java™ 2 Platform, Enterprise Edition, 1999. Disponível em <http://java.sun.com/j2ee/reference/whitepapers/j2ee_guide.pdf>. Acesso: 13 novembro 2008

_____. The Java™ EE 5 Tutorial, 2006. Disponível em <<http://java.sun.com/javaee/5/docs/tutorial/doc/>>. Acesso: 15 novembro 2008.