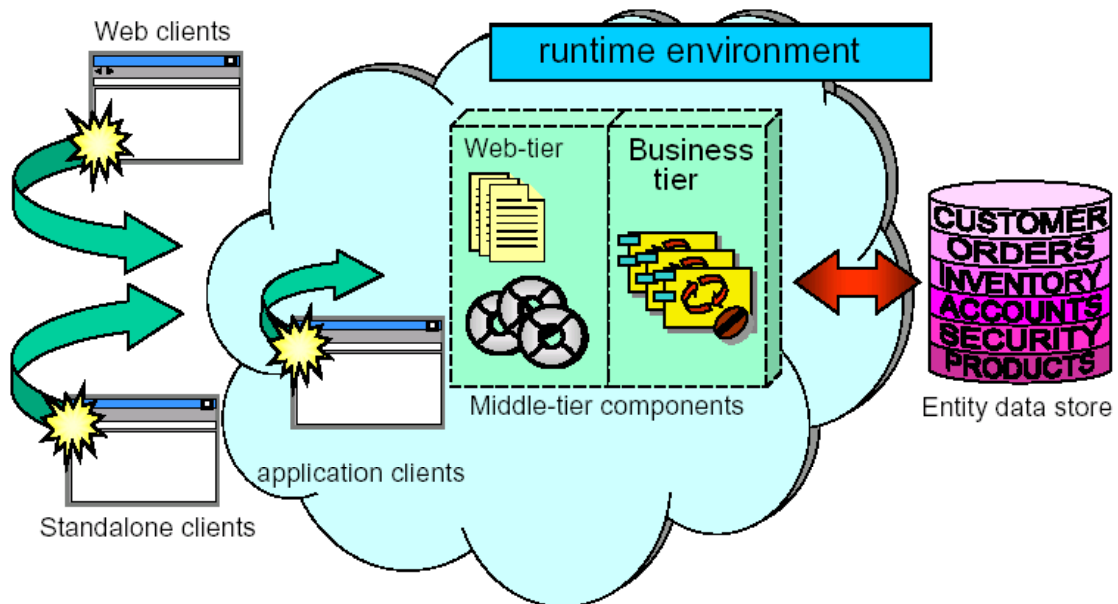




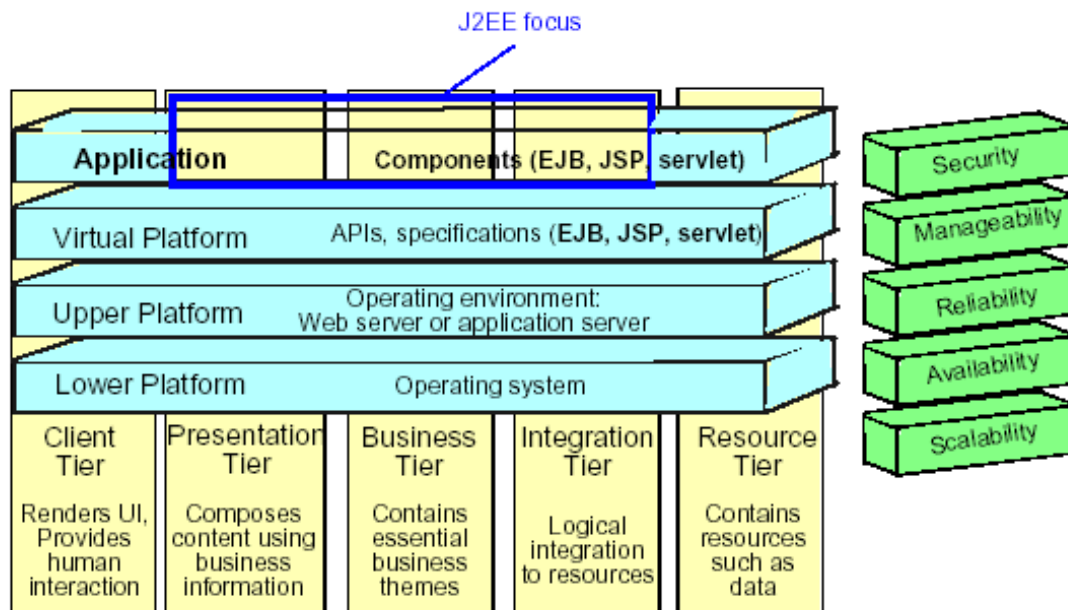
Administração e Manutenção do JBoss Application Server 3.0

1. Introdução aos Servidores de Aplicações

Um servidor de aplicações é uma plataforma de software que disponibiliza serviços padronizados para as aplicações e tem como principal função a gerência de acesso a Banco de Dados e a serviços como eMail, Sistema de Arquivos e integração com outros serviços, e ainda prover o controle das Transações, gerência da Persistência, gerência de Memória e dar suporte a aplicações distribuídas.



Separação das Responsabilidades da Aplicação, Servidor e Sistema Operacional



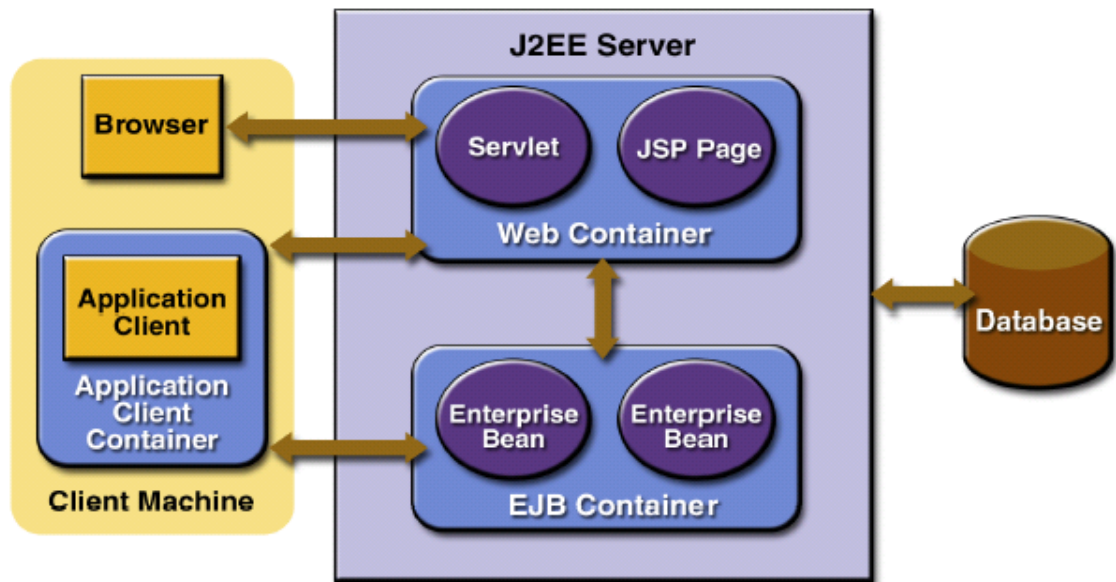
2. Introdução a J2EE.

Um servidor de aplicações J2EE, atualmente segue as especificações da JavaSoft para a Java2 Enterprise Edition 1.3.1. O detalhe desta especificação pode ser encontrado em: <http://java.sun.com/j2ee/1.3/download.html#platformspec>

A J2EE, é uma plataforma de Software, que une várias APIs padronizadas pelo JCP (Java Community Process), e tem como principal objetivo prover um conjunto serviços para que os desenvolvimento de aplicações seja simplificado.

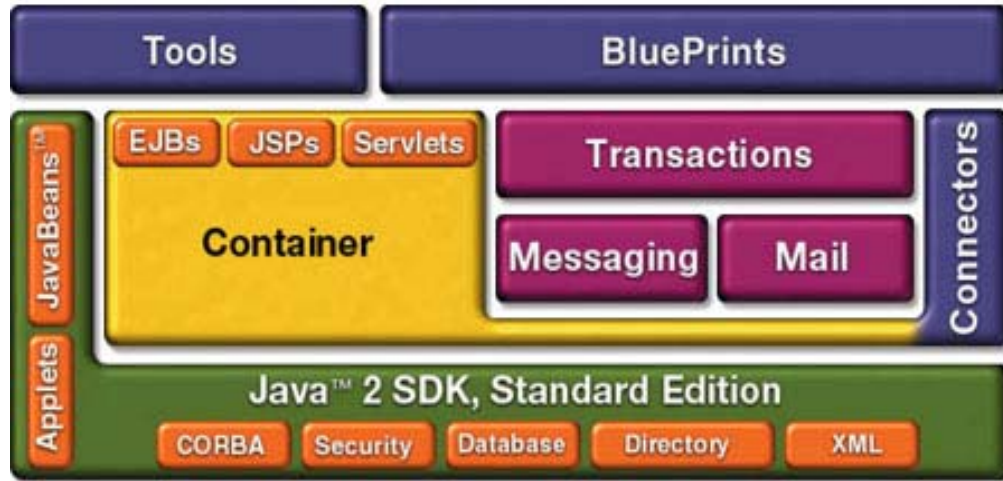
Esta plataforma fica com a responsabilidade de:

- ambiente confiável de execução de aplicativos (Runtime e FailOver);
- ambiente de publicação e atualização de aplicativos (Deployment);
- fornecer interfaces de acesso a recursos e prover a gerência de uso desses recursos (Resource Management);
- fornecer mecanismos de autenticação e autorização (Security Management);
- controlar a concorrência no acesso á componentes da aplicação e garantir o modelo ACID (Transaction Management);
- implementar e suportar a persistência de dados (Persistence Management);
- permitir a distribuição dos componentes em vários ambientes de runtime (Distributed Component Enviromnent);
- conjunto de “BluePrints”, boas práticas e guidelines para implementação;
- ferramentas são criadas e disponibilizadas pelo fabricante ou por terceiros;



Plataforma de Runtime da J2EE

Serviços de Runtime da J2EE



3. Benefícios da J2EE.

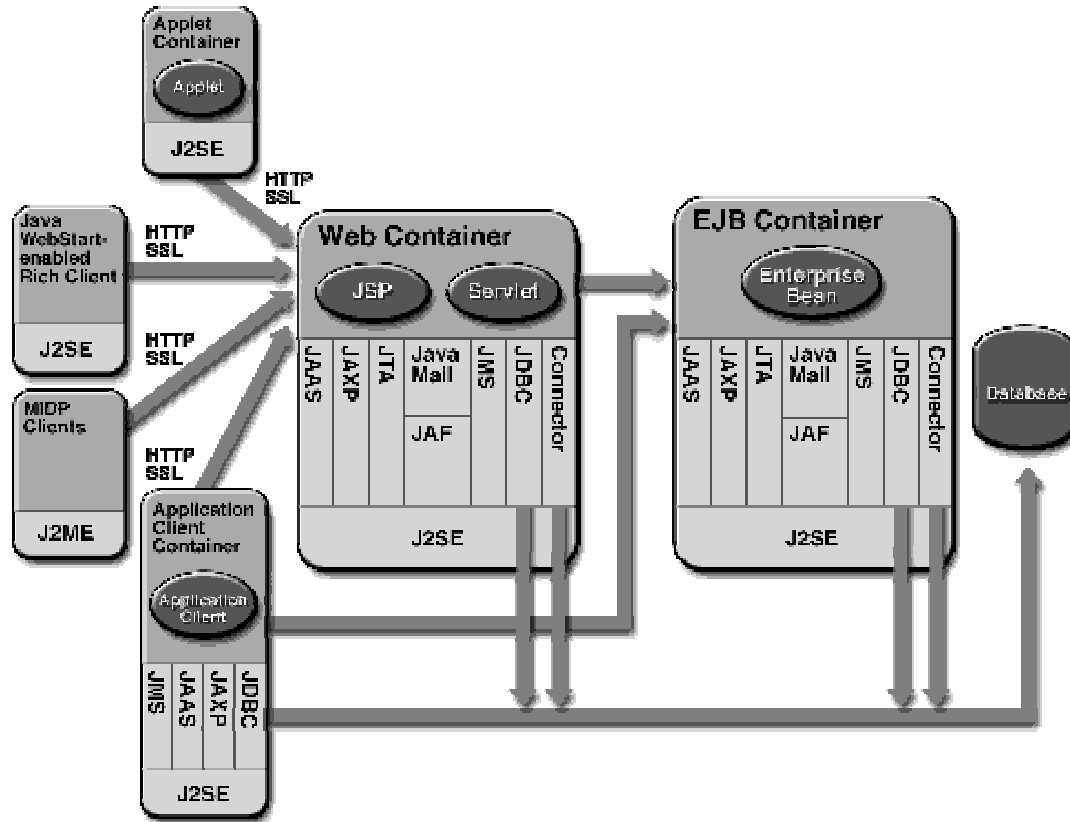
- 3.1. Mais de 20 "vendedores" certificados;
- 3.2. Open Industry Standards (Oracle, IBM, SUN, Compuware, SAP, HP, BEA, Borland, Apache, etc);
- 3.3. Totalmente aderente aos padrões da UML/OOAD e RUP;
- 3.4. Business Component-Based;
- 3.5. Suporte de mais de 40 IDEs;
- 3.6. Portabilidade total das Aplicações, WIN32, WIN64, UNIX, UNIX64, MainFrame, etc.
- 3.7. Garantia do Retorno de Investimento;

4. Principais APIs da J2EE.

J2EE 1.2	J2EE 1.3
Servlets 2.2, JSP 1.1	Servlets 2.3, JSP 1.2
EJB 1.1 (SessionBeans, EntityBeans)	EJB 2.0 (SessionBeans, EntityBeans, MessageBeans)
CMP 1.0 (Container Managed Persistence 1.0)	CMP 2.0 (Container Managed Persistence 2.0) EJBQL (SQL ANSI)
JNDI, JTA/JTS (Transações), DataSource (Pool), IIOP (IDL/RMI), JMS	
JavaMail 1.1 (POP3,SMTP)	JavaMail 1.2 (POP3, SMTP, IMAP)
JDBC 2.0, 2.1	JDBC 2.1, 3.0
JDK 1.2.2 ou 1.3.1	JDK 1.3, JDK 1.4 (HotSpot VM)

5. Java2 Enterprise Edition / Java2 Standard Edition

Plataformas de Software Java



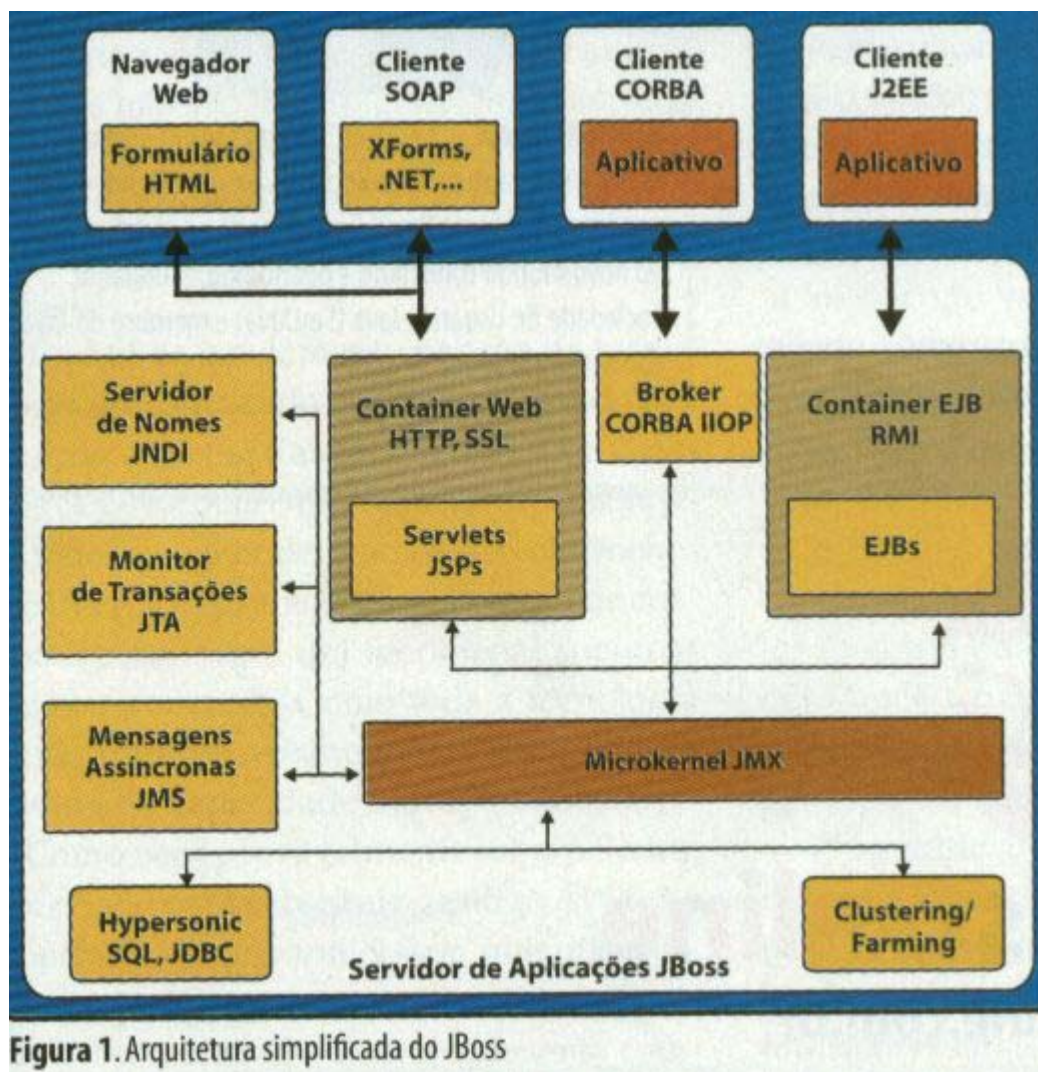
Nesta figura podemos ver como a J2EE foi criada, quais seus serviços, sua dependência da J2SE e sua integração com outras plataformas.

6. Introdução ao JBOSS Application Server 3.0

Totalmente construído em Java, este servidor de aplicações é gratuito e foi desenvolvido por uma comunidade OpenSource, a JBoss.org, e segue um modelo de licença GPL.

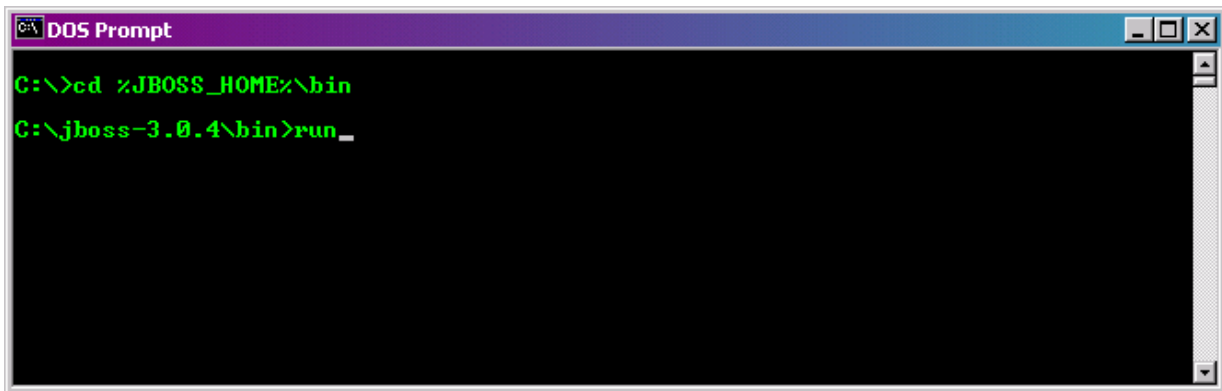
O JBOSS Application Server 3.0, foi implementado seguindo as especificações da J2EE 1.3.1.

Na figura abaixo podemos identificar os principais componentes da sua arquitetura que provê a execução de componentes de Software Java, baseado na plataforma J2EE.



7. Instalação do JBOSS Application Server 3.0

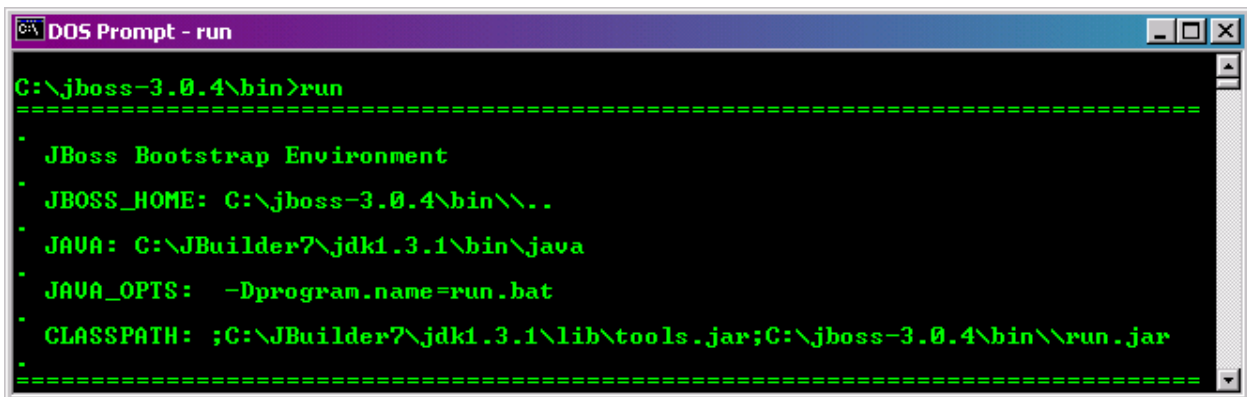
1. Instalando.
 - a. é necessário estar instalada e configurada na máquina que vai rodar o JBOSS 3.0, uma JRE 1.3.1 (Java Runtime Environment) ou superior, ou uma JDK 1.3.1 (Java Development Kit) ou superior;
 - b. Descompactar o JBoss-3.0.4_Tomcat-4.0.6.zip em: c:\jboss-3.0.4
 - c. Configurar as seguintes variáveis de Ambientes:
 - i. JBOSS_HOME = c:\jboss-3.0.4
 - ii. TOMCAT_HOME = c:\jboss-3.0.4
 - iii. CATALINA_HOME = c:\jboss-3.0.4
 - iv. PATH=%JBOSS_HOME%\bin; %PATH%;
 - d. Verificar se o CLASSPATH, PATH e JAVA_HOME estão setados corretamente;
2. Executando.
 - a. Num prompt de comando navegar para o diretório de instalação do JBOSS:
 - b. Executar o run.bat



```

C:\>cd %JBOSS_HOME%\bin
C:\jboss-3.0.4\bin>run
    
```

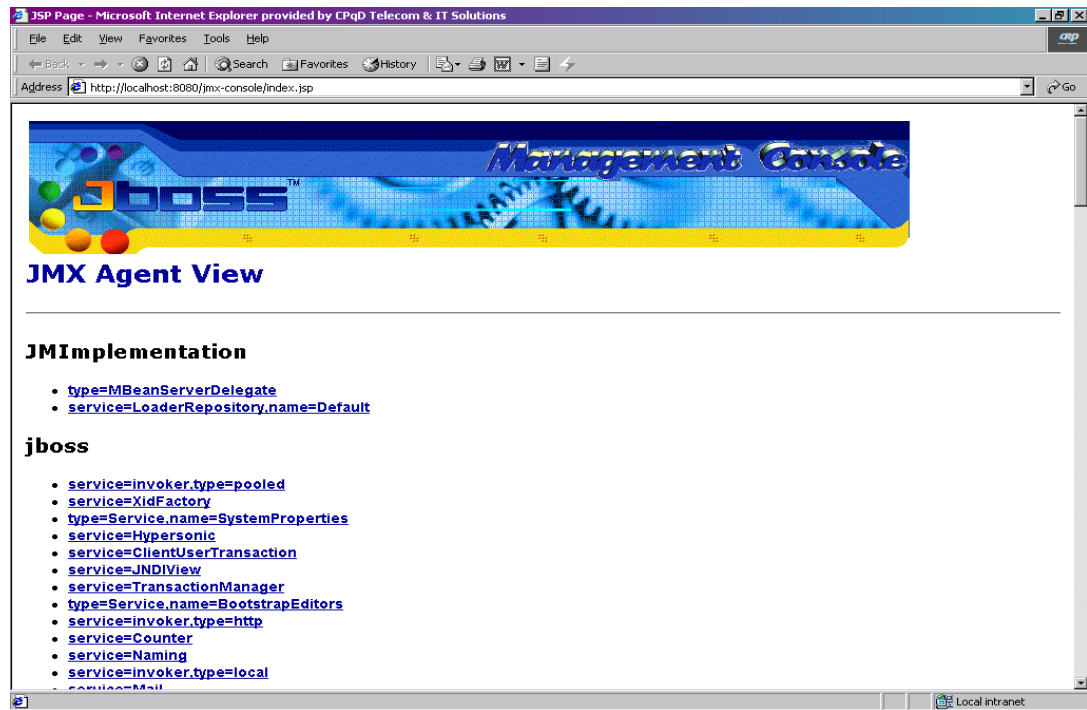
- c. Deve acontecer a carga do JBOSS 3.0



```

C:\jboss-3.0.4\bin>run
=====
* JBoss Bootstrap Environment
* JBOSS_HOME: C:\jboss-3.0.4\bin\..
* JAVA: C:\JBUILDER7\jdk1.3.1\bin\java
* JAVA_OPTS: -Dprogram.name=run.bat
* CLASSPATH: ;C:\JBUILDER7\jdk1.3.1\lib\tools.jar;C:\jboss-3.0.4\bin\run.jar
*
=====
    
```

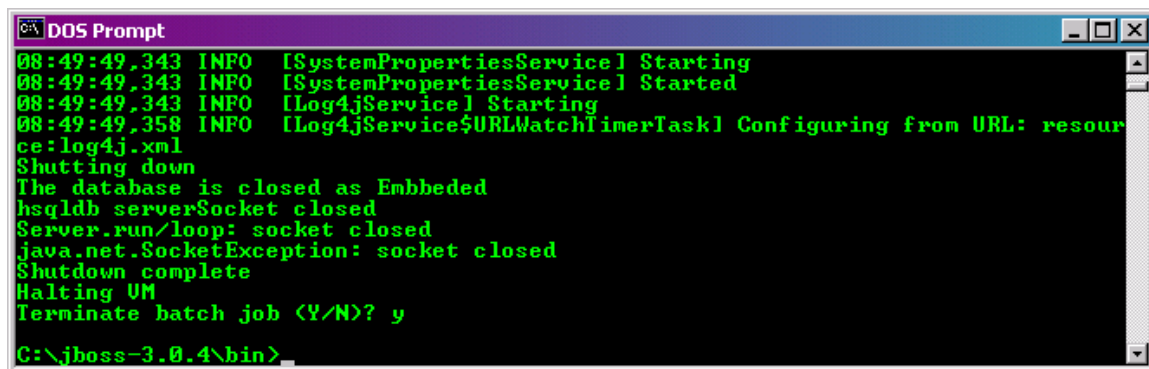
3. Testando a instalação:
 - a. No navegador web, apontar para <http://localhost:8080/jmx-console>



Esta aplicação, representa a console do JMX (Java Management eXtensions) do Servidor de Aplicações JBOSS 3.0. Aqui podemos gerenciar os serviços do WebContainer, do EJBContainer e dos Serviços do J2EE Server.

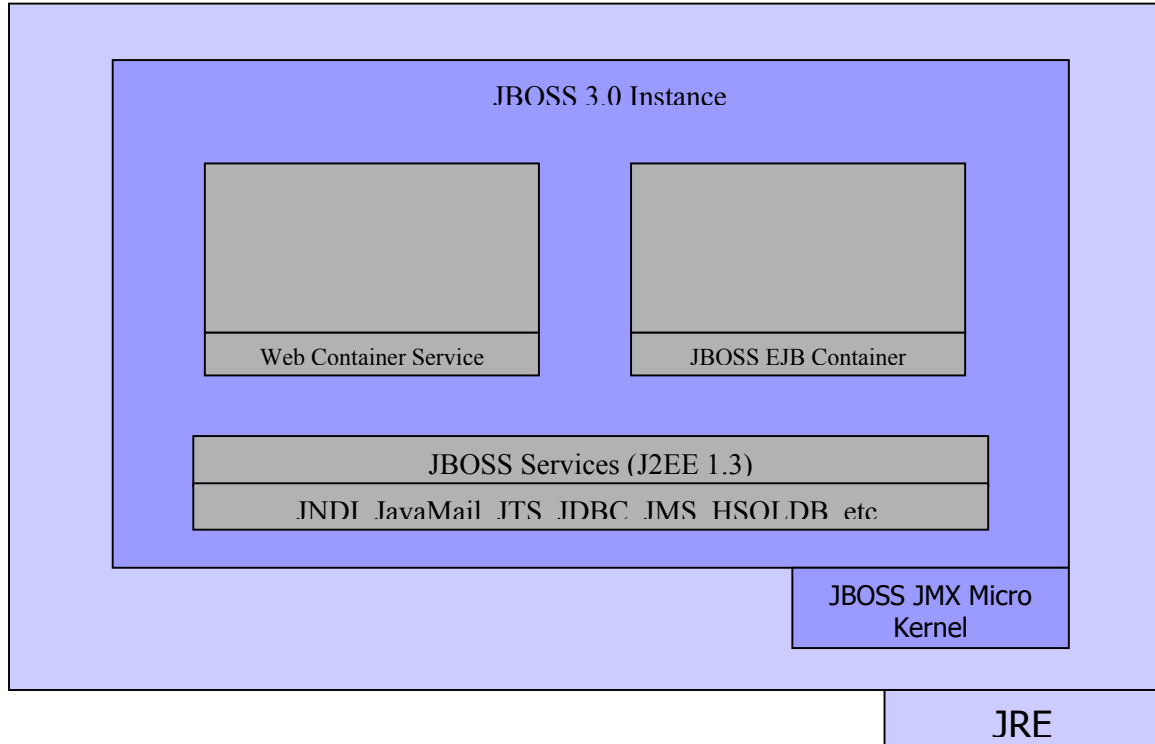
* O JMX foi implementado na versão 3.0, mas não faz parte de J2EE 1.3, ou seja, nem todos os servidores de aplicações baseados na J2EE 1.3.1 Specs, implementam essa console.

4. Parando o Servidor
 - a. Basta teclar {CTRL+C} na janela que está executando ou executar:
%JBOSS_HOME%\bin\shutdown.bat



8. Arquitetura Interna do JBOSS 3.0

A JBOSS.org na construção do servidor optou por uma arquitetura mais simples e funcional, evitando componentes nativos e garantindo a portabilidade.



O JBOSS 3.0 tem como grande benefício a característica de se poder customizar, ou até substituir alguns de seus serviços internos por outros serviços de outros desenvolvedores ou fabricantes de software.

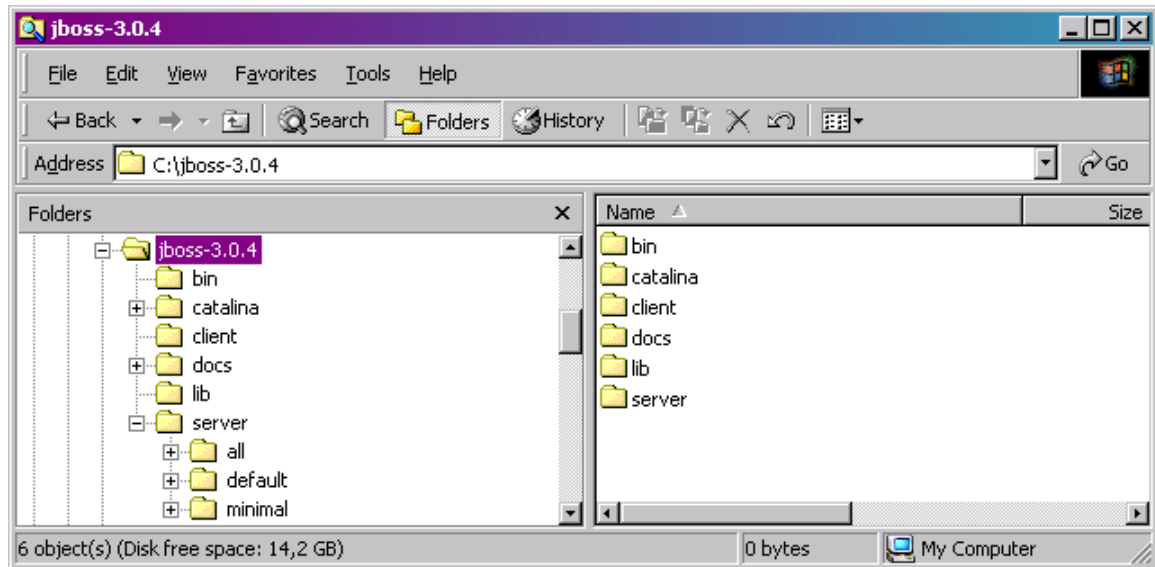
A customização mais comum é do WebContainer. Atualmente o JBOSS tem duas distribuições com WebContainers diferentes:

- 8.1. JBoos 3.0 com Tomcat 4.X – Usa o Jakarta Tomcat 4.X da Apache Software como Engine para a execução de Servlets e JSP.
- 8.2. JBoos 3.0 com JBossWeb– Usa o Jetty 4.0 da JBOSS.org como Engine para a execução de Servlets e JSP.

Basicamente, as diferença entre os WebContainers seriam somente em relação às suas configurações, visto que a sua arquitetura interna é bem semelhante, o que não causaria diferenças gritantes de performance.

9. Configurações do JBOSS 3.0

Durante o processo de carga do MicroKernel, o JBOSS 3.0 verifica dentro do diretório %JBOSS_HOME% se existe a seguinte estrutura:



Dentro do diretório %JBOSS_HOME%\server; ficam as “instâncias”, ou seja, cada diretório, mantém um conjunto de serviços configurados e que caracteriza um instância do servidor.

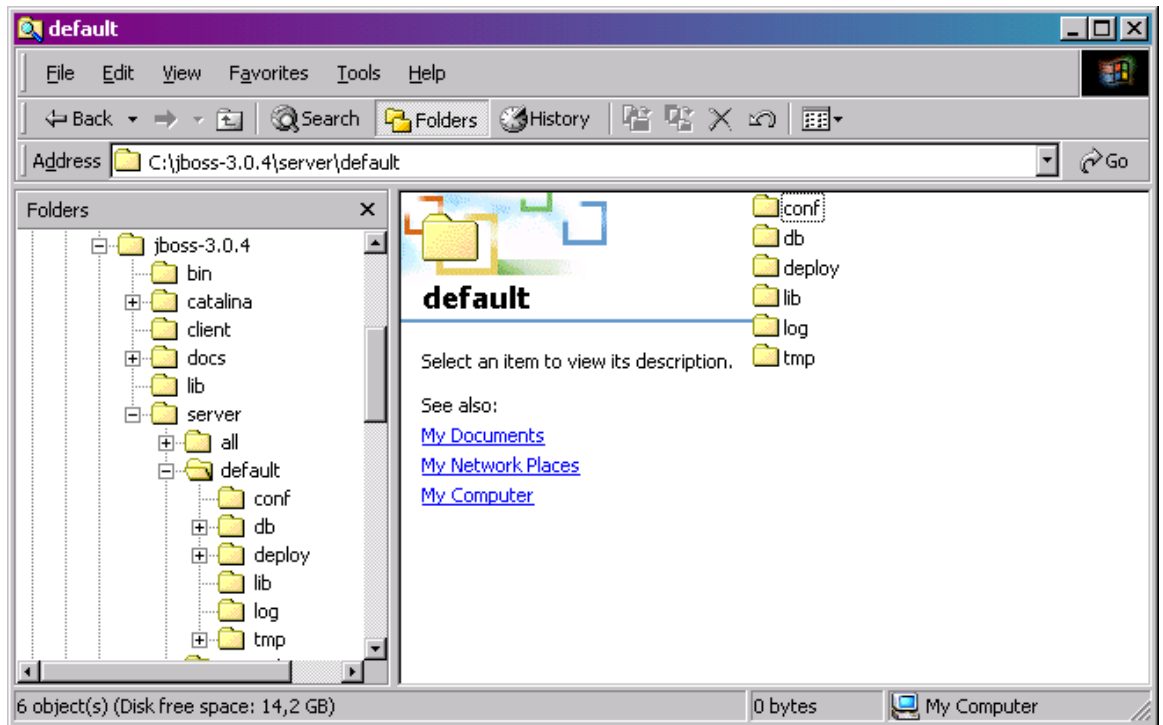
Em runtime, somente existirá por uma instância por JRE.

Durante o processo de “StartUp”, a ordem de busca é: default, all e minimum. E na chamada do executor do startup do servidor, pode-se passar um opção para se escolher uma determinada instância. Ex:

```

C:\>DOS Prompt - run
08:39:42,532 INFO [Server] JBoss Release: JBoss-3.0.4 CUSTag=JBoss_3_0_4
08:39:42,532 INFO [Server] Home Dir: C:\jboss-3.0.4
08:39:42,532 INFO [Server] Home URL: file:/C:/jboss-3.0.4/
08:39:42,532 INFO [Server] Library URL: file:/C:/jboss-3.0.4/lib/
08:39:42,532 INFO [Server] Patch URL: null
08:39:42,532 INFO [Server] Server Name: default
08:39:42,548 INFO [Server] Server Home Dir: C:\jboss-3.0.4\server\default
08:39:42,548 INFO [Server] Server Home URL: file:/C:/jboss-3.0.4/server/default/
08:39:42,548 INFO [Server] Server Data Dir: C:\jboss-3.0.4\server\default\db
08:39:42,579 INFO [Server] Server Temp Dir: C:\jboss-3.0.4\server\default\tmp
  
```

Dessa forma o JBOSS JMX MicroKernel carregou as configurações da instância “default” e que estão representadas no seguinte diretório:



Descrição dos diretórios, válida para qualquer instância:

\conf – configuração do JBOSS MicroKernel, EJB Container, JAAS e JMS Broker e JTS (Transações). Se for alterado algum de configuração deste diretório é necessário o restart do servidor.

\db – dados temporários usados em runtime;

\deploy – diretório onde se configura os serviços de Container Web, JMS Destinations, Clusters, JavaMail, JDBC DataSources, Conectores JCA, Conectores para IIOP, JNDI, Scheduler*, serviços customizados* e “deploy” dos componentes de aplicações. Se for alterado algum de configuração deste diretório ou for feito um novo deploy, ou redeploy de uma aplicação, não é necessário o restart do servidor.

\lib – diretório usado para o “deploy” de componentes de terceiros, pacotes de classes em formato .jar;

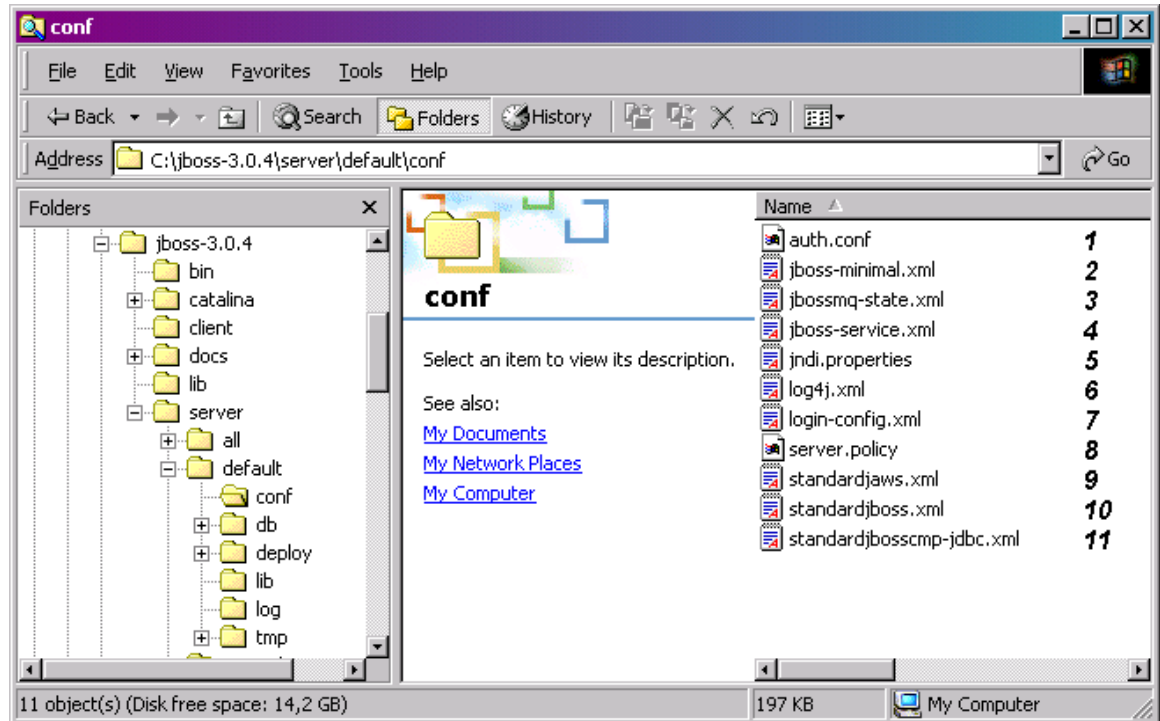
\log – diretório onde serão colocados os logs do servidor;

\tmp – diretório usando em runtime como temporário de deploy de componentes e aplicações.

9.1. Os arquivos de configurações do MicroKernel.

Cada arquivo de configuração de um determinado serviço é um arquivo no formato “xml” ou no formato “properties” e segue um formato específico para cada tipo de serviço.

Explorando o diretório “conf”, temos:



Descrição dos arquivos de configurações:

- 1 – dos módulos do JAAS;
- 2 – do JBOSS MicroKernel – instancia com um mínimo de serviços;
- 3 – da autenticação de acesso ao Broker do JBOSS JMS;
- 4 – do JBOSS MicroKernel – instancia com os serviços da J2EE 1.3;
- 5 – das classes de contexto da JNDI;
- 6 – do Log4J;
- 7 – dos repositórios de usuários para o JAAS;
- 8 – política de segurança do JBOSS para a JRE;
- 9 – do Java Web Start;
- 10 e 11 – do JBOSS MicroKernel para o EJB Container;

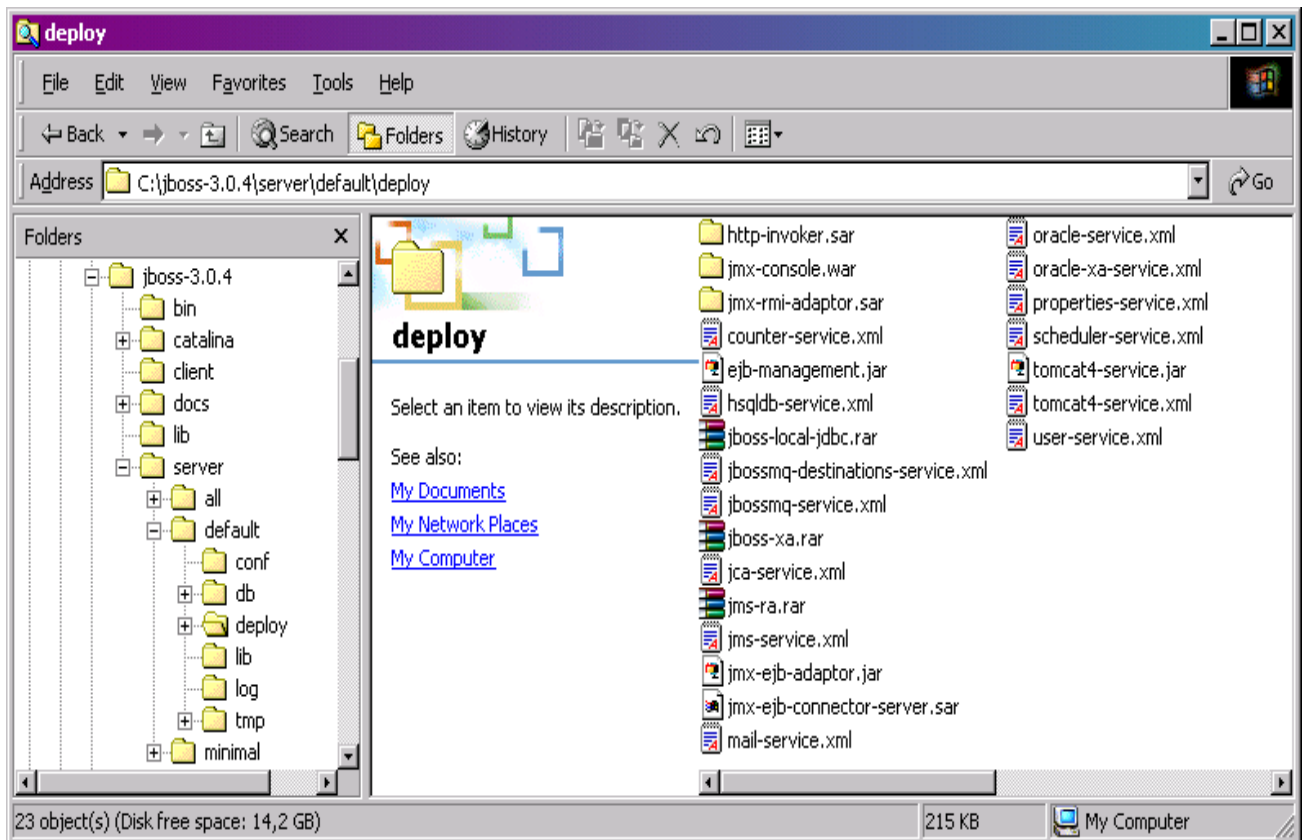
Seria interessante você investir um tempo em abrir cada um dos arquivos para facilitar o entendimento de cada um deles. Dentro de cada arquivo, existem sempre comentários e modelos de como fazer novas configurações.

9.2. Os arquivos de configurações dos Serviços.

Também como os arquivos do MicroKernel, as configurações dos Serviços da J2EE 1.3, bem como as integrações dos WebContainers, estão em arquivos no formato xml, e devem seguir a seguinte nomenclatura:

xxxxxx-service.xml, é por essa nomenclatura, que o MicroKernel sabe que esse arquivo configura um novo serviço para as aplicações.

Explorando o diretório “deploy”, temos:



Seria interessante você investir um tempo em abrir cada um dos arquivos para facilitar o entendimento de cada um deles. Dentro de cada arquivo, existem sempre comentários e modelos de como fazer novas configurações.

9.2.1. Explorando os principais arquivos de configurações dos Serviços.

Examinando o arquivo **tomcat4-service.xml**, vemos que ele é um arquivo de configuração de serviço. E seu conteúdo descreve as configurações do Tomcat 4.0 para o JMX MicroKernel (Mbean) poder iniciar o serviço e configurá-lo adequadamente.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE server [
  <!ENTITY catalina.home "../catalina">
]>
<server>
  <classpath codebase="file:&catalina.home;/common/lib/" archives="*" />
  <classpath codebase="file:&catalina.home;/server/lib/" archives="*" />
  <classpath codebase="file:&catalina.home;/bin/" archives="*" />
  <classpath codebase="file:&catalina.home;/lib/" archives="*" />
  <classpath codebase="." archives="tomcat4-service.jar" />
  <mbean code="org.jboss.web.catalina.EmbeddedCatalinaServiceSX"
    name="jboss.web:service=EmbeddedCatalinaSX">
    <attribute name="CatalinaHome">&catalina.home;</attribute>
    <!-- Uncomment this if you want interval snapshot for the
         session clustering.
    <attribute name="SnapshotMode">interval</attribute>
    <attribute name="SnapshotInterval">2000</attribute>
    -->
    <attribute name="Config">
      <Server>
        <Service name = "JBoss-Tomcat">
          <Engine name="MainEngine" defaultHost="localhost">
            <Logger className = "org.jboss.web.catalina.Log4jLogger"
              verbosityLevel = "error" category = "org.jboss.web.localhost.Engine"/>
            <Host name="cpqd050359" unpackWARS="true" appBase="deploy">
              <Valve className = "org.apache.catalina.valves.AccessLogValve"
                prefix = "localhost_tomcat_access" suffix = ".log"
                pattern = "common" directory = "../server/default/log" />
            <DefaultContext cookies = "true" crossContext = "true" override = "true" />
          </Host>
        </Engine>
        <!-- A HTTP Connector on port 8080 -->
        <Connector className = "org.apache.catalina.connector.http.HttpConnector"
          port = "8080" minProcessors = "5" maxProcessors = "75"
          enableLookups = "false" acceptCount = "10" debug = "0"
          connectionTimeout = "60000"/>

        <!-- Define an AJP 1.3 Connector on port 8009 -->
        <Connector className="org.apache ajp.tomcat4.Ajp13Connector"
          port="8009" minProcessors="5" maxProcessors="75" enableLookups = "false"
          acceptCount="10" debug="0" connectionTimeout = "60000"/>

      </Service>
    </Server>
  </attribute>
</mbean>
</server>
```

Para o JBOSS, o importante é a TAG MBean e suas dependências.

Para entender as configurações do Tomcat 4, é necessário consultar a documentação dele em: <http://jakarta.apache.org/tomcat/tomcat-4.0-doc/config/index.html>

Toda vez que configuramos um serviços, vamos precisar de duas referências, uma do JBOSS, para configurarmos o MicroKernel e outra do referência do serviço que estamos configurando.

Este procedimento garante a independência das configurações.

Temos que lembrar que um DataSource é um serviço da J2EE, e dentro deste arquivo existem configurações desde o nome do Driver do DataSource, passando pelo ConnectionPool e até os Módulos do JAAS para autenticação de usuários.

Ou seja, este MBean é bem complexo e sua configuração deve ser feita com muita atenção.

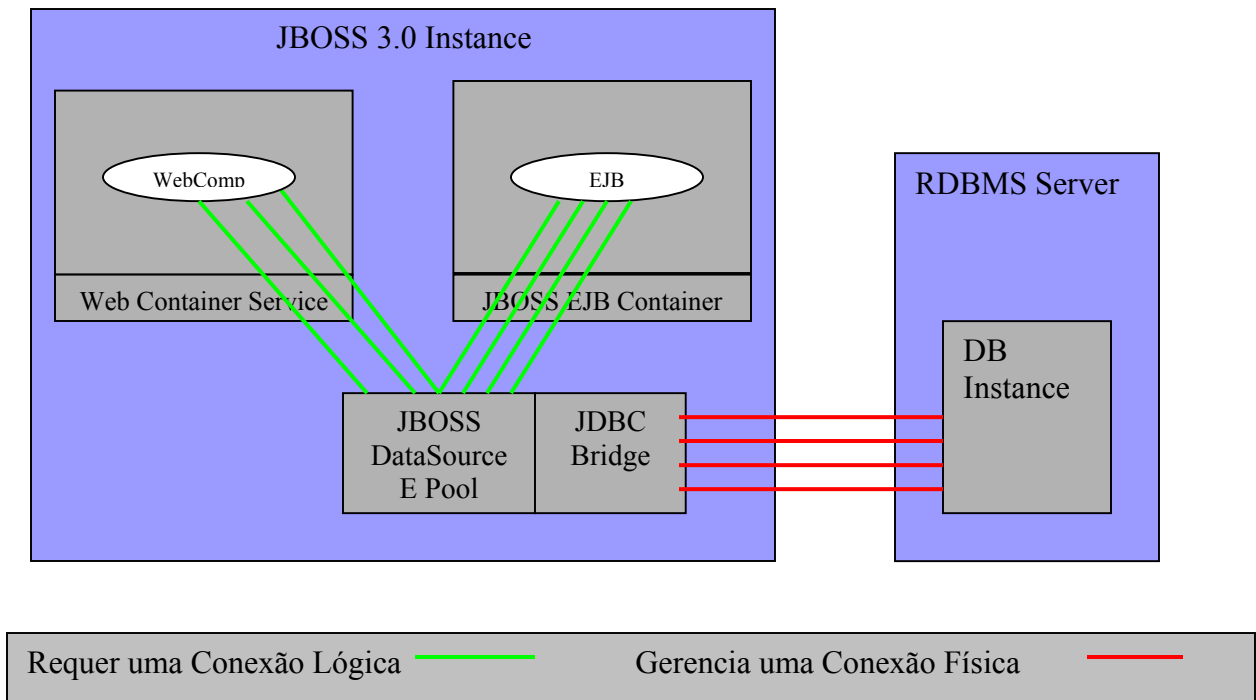
Para entendermos TODOS os arquivos , suas configurações básicas e variações, somente fazendo vários testes com cada uma delas.

Por isso, sugiro que sejam examinados todos os arquivos de configuração que estão dentro do diretório “deploy”.

10. Acessando Bancos de Dados com o JBOSS 3.0

Toda que lidamos com bancos de dados, temos que nos preocupar com alguns pontos, principalmente no assunto que refere o uso de recursos do banco, como conexões, cursores, memória e etc.

O JBOSS não tem como interferir nas consultas que os componentes das aplicações fazem ao banco. Entretanto ele auxilia na gerência de recursos que essas consultas utilizam;



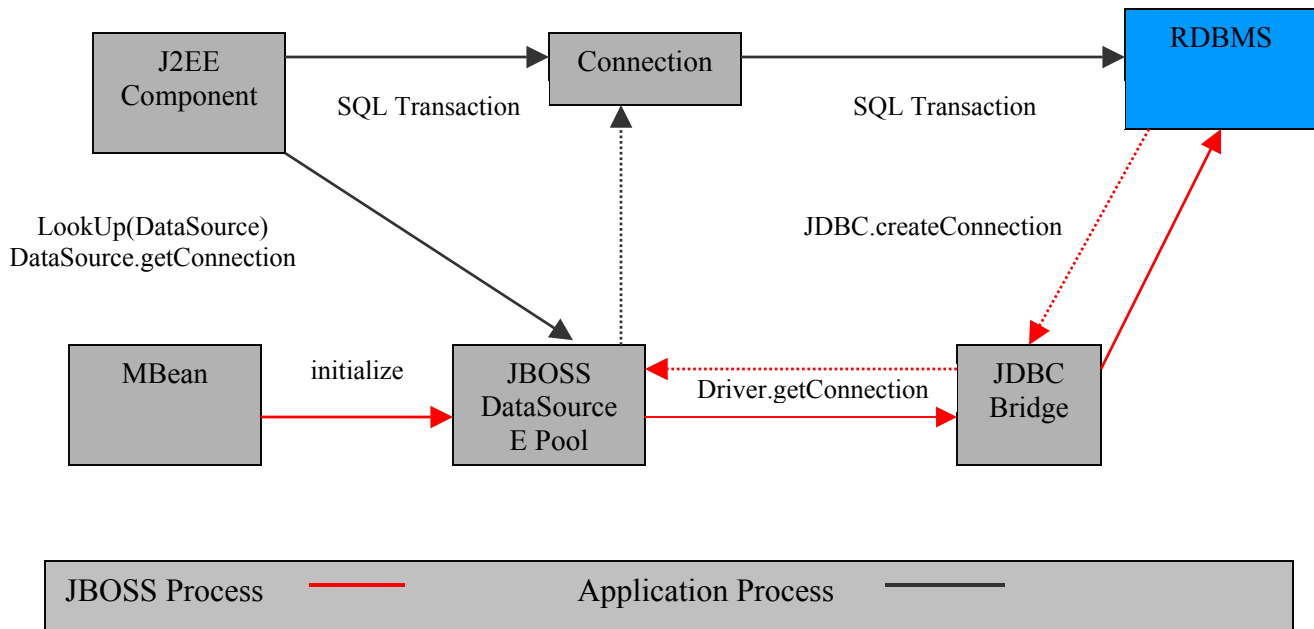
Como podemos ver na figura acima, a responsabilidade do DataSource é criar as Conexões Físicas, colocando-as num Pool, que será gerenciado de acordo com a necessidade de uso de acesso ao bando de dados pelos componentes da aplicação.

Cada DataSource aponta para um único banco de dados. Podemos ter vários DataSources apontando para o mesmo banco de dados, mas nunca poderemo ter um DataSource apontando para vários bancos.

No caso da aplicação necessitar de que numa mesma transação, sejam usados dois DataSources diferentes, deve-se configurar um DataSource que seja capaz de realizar transações distribuidas usando o modelo XA.

E para isso, tanto com o Banco de Dados, como o JDBC Driver devem suporta tal modelo de transações.

Antes de aprendermos a configurar um DataSource, vamos entender como funciona o Pool de Conexões do JBOSS e como instalar os Drivers JDBC para os bancos de dados.



O DataSource usa um algoritmo otimizado para controlar o uso das Conexões com o Banco de Dados.

Quando Inicializado, ele cria as conexões físicas com o Banco de Dados, autenticando-as com o usuário que está configurado para aquele DataSource, e aloca-as num Pool.

No momento em que a aplicação requer uma conexão, ele marca-a como “em uso” e dispara um “timer” para controlar o tempo de uso.

Quando a aplicação termina a transação, ela fecha a conexão, entretanto, o DataSource intercepta a chamada, e a marca como “livre”, e assim ele controla as conexões.

Se por algum motivo, todas as conexões físicas estiverem em uso, ele vai colocando as requisições de conexões numa fila de espera, com o intuito de não negar acesso ao serviço do Banco de Dados, mas sim gerenciá-lo.

Se o tempo de uso for superior ao configurado no DataSource, ele vai fechar a conexão, não importando seu status, para poder criar uma nova e colocá-la no Pool para ser usada por outra requisição.

Uma forma de medir se ao tamanho “mínimo” e “máximo” do Pool são suficientes ou se sua configuração de “timeout” e “blocking timeout” estão razoáveis, é analisando os Logs e identificado se as conexões estão sendo fechadas e recriadas pelo JBOSS, no caso de o tempo de uso estar baixo, ou se o tamanho do pool não é suficiente pela carga de usuários.

10.1. Configurando um DataSource non-XA para o HyperSonic

Examinando o arquivo **hsqldb-service.xml**, vemos que ele é um arquivo de configuração de serviço. Configura os recurso de um DataSource (non-XA) para um banco de dados chamado HyperSonic.

```
<?xml version="1.0" encoding="UTF-8"?>
  <!-- New ConnectionManager setup for default hsql dbs -->
  <mbean code="org.jboss.resource.connectionmanager.LocalTxConnectionManager" ]
    name="jboss.jca:service=LocalTxCM,name=hsqldbDS">
    <!-- Include a login module configuration named HsqlDbRealm.
      Update your login-conf.xml, here is an example for a
      ConfiguredIdentityLoginModule:
    <application-policy name = "HsqlDbRealm">
      <authentication>
        <login-module code =
"org.jboss.resource.security.ConfiguredIdentityLoginModule" flag = "required">
          <module-option name = "principal">sa</module-option>
          <module-option name = "userName">sa</module-option>
          <module-option name = "password"></module-option>
          <module-option name =
"managedConnectionFactoryName">jboss.jca:service=LocalTxCM,name=hsqldbDS</modul
e-option>
        </login-module>
      </authentication>
    </application-policy>
    -->
    <!--uncomment out this line if you are using the DB2DbRealm above
    <attribute name="SecurityDomainJndiName">HsqlDbRealm</attribute>
    -->
    <depends optional-attribute-name="ManagedConnectionFactoryName">
      <mbean code="org.jboss.resource.connectionmanager.RARDeployment"
name="jboss.jca:service=LocalTxDS,name=hsqldbDS">
        <attribute name="JndiName">DefaultDS</attribute>
        <attribute name="ManagedConnectionFactoryProperties">
          <properties>
            <config-property name="ConnectionURL"
type="java.lang.String">jdbc:hsqldb:hsql://localhost:1476</config-property>
            <config-property name="DriverClass"
type="java.lang.String">org.hsqldb.jdbcDriver</config-property>
            <config-property name="UserName"
type="java.lang.String">sa</config-property>
            <config-property name="Password" type="java.lang.String"></config-
property>
          </properties>
        </attribute>
        <!--Below here are advanced properties -->
        <depends optional-attribute-
name="OldRarDeployment">jboss.jca:service=RARDeployment,name=JBoss
LocalTransaction JDBC Wrapper</depends>
        <depends>jboss:service=HyperSonic</depends>
      </mbean>
    </depends>
```

```

    <depends optional-attribute-name="ManagedConnectionPool">
      <mbean
code="org.jboss.resource.connectionmanager.JBossManagedConnectionPool"
name="jboss.jca:service=LocalTxPool,name=hsqldbDS">
    <attribute name="MinSize">0</attribute>
    <attribute name="MaxSize">50</attribute>
    <attribute name="BlockingTimeoutMillis">5000</attribute>
    <attribute name="IdleTimeoutMinutes">15</attribute>
    <!--criteria indicates if Subject (from security domain) or app
supplied
distinguish      parameters (such as from getConnection(user, pw)) are used to
connections in the pool. Choices are
ByContainerAndApplication (use both),
ByContainer (use Subject),
ByApplication (use app supplied params only),
ByNothing (all connections are equivalent, usually if adapter
supports      reauthentication)-->
    <attribute name="Criteria">ByContainer</attribute>
  </mbean>
</depends>
<depends optional-attribute-
name="CachedConnectionManager">jboss.jca:service=CachedConnectionManager</depen
ds>
  <depends optional-attribute-
name="JaasSecurityManagerService">jboss.security:service=JaasSecurityManager</d
epends>
  <attribute name="TransactionManager">java:/TransactionManager</attribute>
  <!--make the rar deploy! hack till better deployment-->
  <depends>jboss.jca:service=RARDeployer</depends>
</mbean>
<!-- Moved to end to test anonymous depends -->
<mbean code="org.jboss.jdbc.HypersonicDatabase"
  name="jboss:service=Hypersonic">
  <attribute name="Port">1476</attribute>
  <attribute name="Silent">true</attribute>
  <attribute name="Database">default</attribute>
  <attribute name="Trace">>false</attribute>
</mbean>
</server>

```

10.2. Configurando um DataSource XA para o Oracle

Examinando o arquivo **oracle-xa-service.xml**, vemos que ele é um arquivo de configuração de serviço. Configura os recurso de um DataSource (XA) para um banco de dados Oracle.

```
<?xml version="1.0" encoding="UTF-8"?>
<server>
  <!--
===== --
>
  <!-- ConnectionManager setup for xa oracle dbs
-->
  <!-- YOU MUST CHANGE THE XidFactoryMBean config (in conf/jboss-
service.xml -->
  <!-- or transaction-service.xml) to this: -->
  <mbean code="org.jboss.tm.XidFactory"
    name="jboss:service=XidFactory">
    <attribute name="Pad">true</attribute>
  </mbean>
  -->
  <!--
===== --
>
  <mbean
code="org.jboss.resource.connectionmanager.XATxConnectionManager"
    name="jboss:jca:service=XATxCM,name=XAOracleDS">
    <depends>jboss:jca:service=RARDeployer</depends>
    <depends optional-attribute-name="ManagedConnectionFactoryName">
      <mbean code="org.jboss.resource.connectionmanager.RARDeployment"
        name="jboss:jca:service=XATxDS,name=XAOracleDS">
        <depends optional-attribute-name="OldRarDeployment">
          jboss:jca:service=RARDeployment,name=Minerva JDBC
XATransaction ResourceAdapter</depends>
        <attribute name="ManagedConnectionFactoryProperties">
          <properties>
            <config-property name="XADataSourceProperties"

type="java.lang.String">URL=jdbc:oracle:thin:@baleia:1527:bal817i1</con
fig-property>
            <config-property name="XADataSourceClass"

type="java.lang.String">oracle.jdbc.xa.client.OracleXADataSource</confi
g-property>
            <config-property name="UserName"
              type="java.lang.String">ebpp01des</config-property>
            <config-property name="Password"
              type="java.lang.String">ebpp01des</config-property>
          </properties>
        </attribute>
        <attribute name="JndiName">XAOracleDS</attribute>
      </mbean>
    </depends>
    <depends optional-attribute-name="ManagedConnectionPool">
```



```

    <mbean
code="org.jboss.resource.connectionmanager.JBossManagedConnectionPool"
    name="jboss.jca:service=XATxPool,name=XAOracleDS">
    <attribute name="MinSize">5</attribute>
    <attribute name="MaxSize">10</attribute>
    <attribute name="BlockingTimeoutMillis">1000</attribute>
    <attribute name="IdleTimeoutMinutes">15</attribute>
    <attribute name="Criteria">ByContainer</attribute>
    </mbean>
</depends>
<depends optional-attribute-
name="CachedConnectionManager">jboss.jca:service=CachedConnectionManage
r</depends>

    <attribute name="SecurityDomainJndiName">OracleDbRealm</attribute>

    <depends optional-attribute-
name="JaasSecurityManagerService">jboss.security:service=JaasSecurityMa
nager</depends>

    <attribute
name="TransactionManager">java:/TransactionManager</attribute>
    </mbean>
</server>

```

Inserir no **conflogin-config.xml** as seguintes linhas:

```

    <application-policy name = "OracleDbRealm">
    <authentication>
    <login-module code =
"org.jboss.resource.security.ConfiguredIdentityLoginModule"
    flag = "required">
    <module-option name = "principal">ebpp01des</module-option>
    <module-option name = "userName">ebpp01des</module-option>
    <module-option name = "password">ebpp01des</module-option>
    <module-option name =
"managedConnectionFactoryName">jboss.jca:service=XATxCM,name=XAOracleDS</module
-option>
    </login-module>
    </authentication>
    </application-policy>

```

Alterar no arquivo **confjboss-service.xml** as seguintes linhas:

```

<!-- ===== -->
<!-- Transactions -->
<!-- ===== -->
<mbean code="org.jboss.tm.XidFactory"
    name="jboss:service=XidFactory">
    <attribute name="Pad">true</attribute>
</mbean>

```

10.3. Configurando um DataSource non-XA para o MySQL

```
<?xml version="1.0" encoding="UTF-8"?>

<server>
  <mbean
code="org.jboss.resource.connectionmanager.LocalTxConnectionManager"
    name="jboss.jca:service=LocalTxCM,name=mysqlLDS">

    <depends optional-attribute-name="ManagedConnectionFactoryName">
      <!--embedded mbean-->
      <mbean code="org.jboss.resource.connectionmanager.RARDeployment"
name="jboss.jca:service=LocalTxDS,name=mysqlLDS">

        <attribute name="JndiName">mysqlLDS</attribute>
        <attribute name="ManagedConnectionFactoryProperties">
          <properties>
            <config-property name="ConnectionURL"
type="java.lang.String">jdbc:mysql://localhost/cursos</config-property>
            <config-property name="DriverClass"
type="java.lang.String">org.gjt.mm.mysql.Driver</config-property>
            <config-property name="UserName"
type="java.lang.String">root</config-property>
            <config-property name="Password"
type="java.lang.String"></config-property>
          </properties>
        </attribute>
        <depends optional-attribute-name="OldRarDeployment">
          jboss.jca:service=RARDeployment,name=JBoss LocalTransaction
JDBC Wrapper</depends>
        </mbean>
      </depends>
      <depends optional-attribute-name="ManagedConnectionPool">
        <mbean
code="org.jboss.resource.connectionmanager.JBossManagedConnectionPool"
          name="jboss.jca:service=LocalTxPool,name=mysqlLDS">
            <attribute name="MinSize">1</attribute>
            <attribute name="MaxSize">10</attribute>
            <attribute name="BlockingTimeoutMillis">5000</attribute>
            <attribute name="IdleTimeoutMinutes">15</attribute>
            <attribute name="Criteria">ByContainer</attribute>
          </mbean>
        </depends>
        <depends optional-attribute-
name="CachedConnectionManager">jboss.jca:service=CachedConnectionManage
r</depends>
        <depends optional-attribute-
name="JaasSecurityManagerService">jboss.security:service=JaasSecurityMa
nager</depends>
        <attribute
name="TransactionManager">java:/TransactionManager</attribute>
        <depends>jboss.jca:service=RARDeployer</depends>
      </mbean>
    </server>
```

10.4. Configurando um DataSource non-XA para o PointBase

```
<?xml version="1.0" encoding="UTF-8"?>
<server>
  <mbean
code="org.jboss.resource.connectionmanager.LocalTxConnectionManager"
  name="jboss.jca:service=LocalTxCM,name=PointBaseDS">
    <depends optional-attribute-name="ManagedConnectionFactoryName">
      <mbean code="org.jboss.resource.connectionmanager.RARDeployment"
        name="jboss.jca:service=LocalTxDS,name=PointBaseDS">
        <attribute name="JndiName">PointBaseDS</attribute>
        <attribute name="ManagedConnectionFactoryProperties">
          <properties>
            <config-property name="ConnectionURL"
type="java.lang.String">jdbc:pointbase:server://localhost/CursoJBOSS
            </config-property>
            <config-property name="DriverClass"
type="java.lang.String">com.pointbase.jdbc.jdbcUniversalDriver
            </config-property>
            <config-property name="UserName"
type="java.lang.String">JBOSS</config-property>
            <config-property name="Password"
type="java.lang.String">JBOSS</config-property>
          </properties>
        </attribute>
        <depends optional-attribute-name="OldRarDeployment">
          jboss.jca:service=RARDeployment,name=JBoss LocalTransaction
JDBC Wrapper</depends>
        </mbean>
      </depends>
    <depends optional-attribute-name="ManagedConnectionPool">
      <mbean
code="org.jboss.resource.connectionmanager.JBossManagedConnectionPool"
        name="jboss.jca:service=LocalTxPool,name=PointBaseDS">
        <attribute name="MinSize">1</attribute>
        <attribute name="MaxSize">10</attribute>
        <attribute name="BlockingTimeoutMillis">5000</attribute>
        <attribute name="IdleTimeoutMinutes">15</attribute>
        <attribute name="Criteria">ByContainer</attribute>
      </mbean>
    </depends>
    <depends optional-attribute-
name="CachedConnectionManager">jboss.jca:service=CachedConnectionManage
r</depends>
    <depends optional-attribute-
name="JaasSecurityManagerService">jboss.security:service=JaasSecurityMa
nager</depends>
    <attribute
name="TransactionManager">java:/TransactionManager</attribute>
    <depends>jboss.jca:service=RARDeployer</depends>
  </mbean>
</server>
```

10.5. Configurando um DataSource non-XA para o Oracle

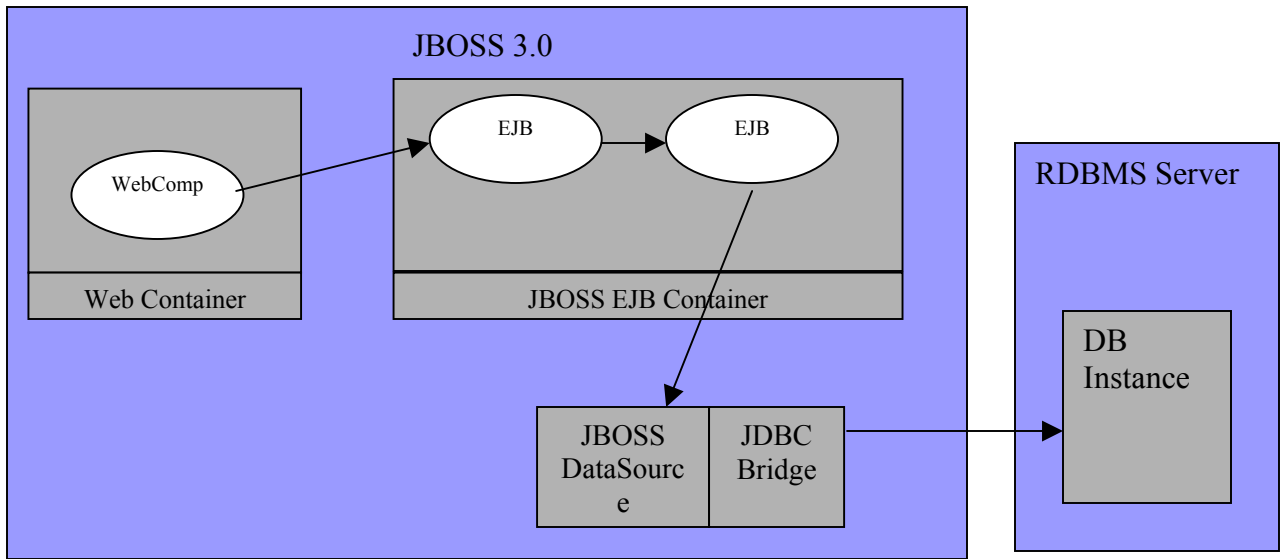
```
<?xml version="1.0" encoding="UTF-8"?>
<server>
  <mbean
code="org.jboss.resource.connectionmanager.LocalTxConnectionManager"
name="jboss.jca:service=LocalTxCM,name=OracleDS">

    <depends optional-attribute-name="ManagedConnectionFactoryName">
      <mbean code="org.jboss.resource.connectionmanager.RARDeployment"
name="jboss.jca:service=LocalTxDS,name=OracleDS">
        <attribute name="JndiName">OracleDS</attribute>
        <attribute name="ManagedConnectionFactoryProperties">
          <properties>
            <config-property name="ConnectionURL"
type="java.lang.String">jdbc:oracle:thin:@baleia:1527:bal817i1
            </config-property>
            <config-property name="DriverClass"
type="java.lang.String">oracle.jdbc.driver.OracleDriver
            </config-property>
            <config-property name="UserName"
type="java.lang.String">ebpp0ldes</config-property>
            <config-property name="Password"
type="java.lang.String">ebpp0ldes</config-property>
          </properties>
        </attribute>
        <depends optional-attribute-
name="OldRarDeployment">jboss.jca:service=RARDeployment,name=JBoss
LocalTransaction JDBC Wrapper</depends>
      </mbean>
    </depends>
    <depends optional-attribute-name="ManagedConnectionPool">
      <mbean
code="org.jboss.resource.connectionmanager.JBossManagedConnectionPool"
name="jboss.jca:service=LocalTxPool,name=OracleDS">
        <attribute name="MinSize">1</attribute>
        <attribute name="MaxSize">10</attribute>
        <attribute name="BlockingTimeoutMillis">5000</attribute>
        <attribute name="IdleTimeoutMinutes">15</attribute>
        <attribute name="Criteria">ByContainer</attribute>
      </mbean>
    </depends>
    <depends optional-attribute-
name="CachedConnectionManager">jboss.jca:service=CachedConnectionManage
r</depends>
    <depends optional-attribute-
name="JaasSecurityManagerService">jboss.security:service=JaasSecurityMa
nager</depends>
    <attribute
name="TransactionManager">java:/TransactionManager</attribute>
    <depends>jboss.jca:service=RARDeployer</depends>
  </mbean>
</server>
```

11. Transações na J2EE

Uma transação consiste em realizar um conjunto de operações dentro do modelo ACID (Atomicidade, Consistência, Isolamento e Durabilidade).

Para isso a J2EE implementa um modelo chamado JTA (Java Transaction Architecture) que serializa o acesso aos objetos envolvidos numa transação e resolve de acordo com o nível de isolamento.



Durante todo o processo desenhado acima, o JBOSS usando a JTA, monitora os eventos em cada Componente EJB, e no caso da transação COMMITAR (COMMIT), o próprio DataSource vai avisar o Banco do COMMIT, e no caso de um excessão de negócio ou erro na aplicação ou falta de recursos, o DataSource vai enviar ao Banco um DESFAZER (ROLLBACK).

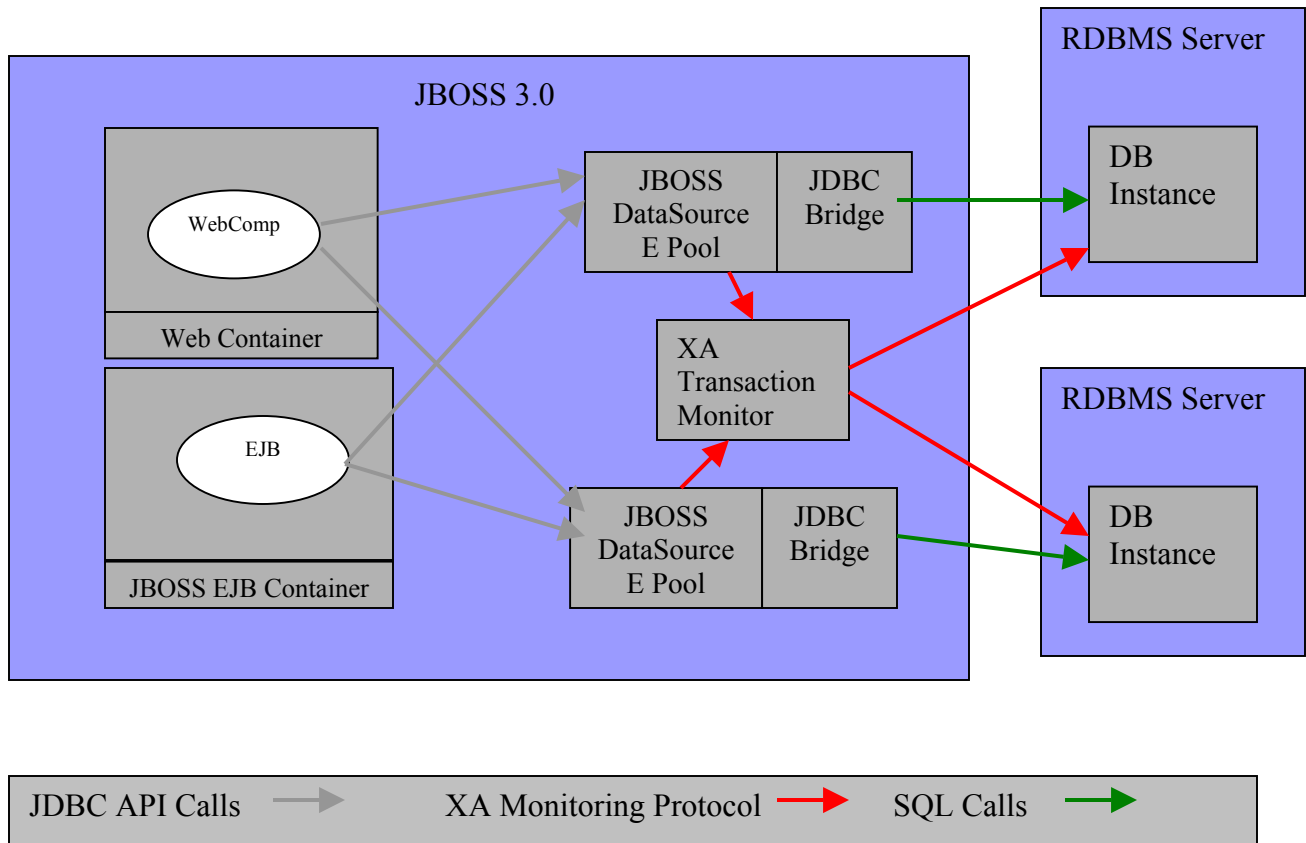
Níveis de isolamento dos métodos dos EJBs:

- NotSupport – Não fará parte do contexto da transação;
- Support – Pode fazer parte de uma transação somente se houve uma;
- Never – Arremessa uma excessão se estiver dentro de uma transação;
- Required – Sempre requer uma transação.
- RequiresNew – Sempre requer uma nova transação;
- Mandatory – Deve fazer de uma transação;

Entender os níveis não é complicado, mas a chamada de métodos entre EJBs deve estar muito bem definida para evitarmos OverHead de atualizações no Banco de Dados.

Cabe aos desenvolvedores demarcar corretamente as transações para evitar esse OverHead.

As transações podem ser distribuídas, ou seja, um mesmo componente J2EE vai executar acessos e requisições de conexões em dois DataSources diferentes. Para que o JBOSS controle o “Two Fase Commit” dos dois DataSources, é necessário que eles estejam configurados como XA.



Se a aplicação por algum motivo a aplicação quiser “defazer” uma das transações num dos DataSources, o monitor XA entra em ação e vai avisar o Banco de Dados que aquela transação deve ser “desfeita” (ROLLBACK), ou no caso da transação acontecer com sucesso, o monitor XA vai avisar que a transação está “completa” (COMMIT) nos dois bancos de dados envolvidos.

Ou seja, os Bancos de Dados só executam o COMMIT ou ROLLBACK se o monitor XA enviar esse comandos.

O monitor XA também é válido para outros recursos do JBOSS, como o JMS ou JavaMail.

Usando-o, eu posso impedir que uma mensagem seja enviada se o acesso a banco de dados falhar. E se o acesso ao banco de dados for feito com sucesso, a mensagem com certeza também será enviada ao JMS ou JavaMail.

12. Deploy de Aplicações J2EE 1.3 no JBOSS 3.0

Como tudo na J2EE é padronizado, o deploy (publicação) de aplicações e componentes também tem um formato e um processo de construção.

Os tipos de componentes da J2EE 1.3 que o JBOSS 3.0 suporta são:

- Servlets – Componentes que são usados para receber e interpretar requisições de um determinado serviço, usando um protocolo definido. Existem basicamente dois tipos de Servlets:
 - GenericServlet – pode ser usado para tratar conexões que usem protocolo TCP para comunicação client-server;
 - HttpServlet – pode ser usado para tratar conexões que usem o protocolo HTTP 1.0 ou 1.1 para comunicação client-server Web.
- Enterprise JavaBeans – Componentes para ambientes distribuídos que são usados para atender requisições de processos de negócios. Suportam transações e podem ou não ser persistentes.

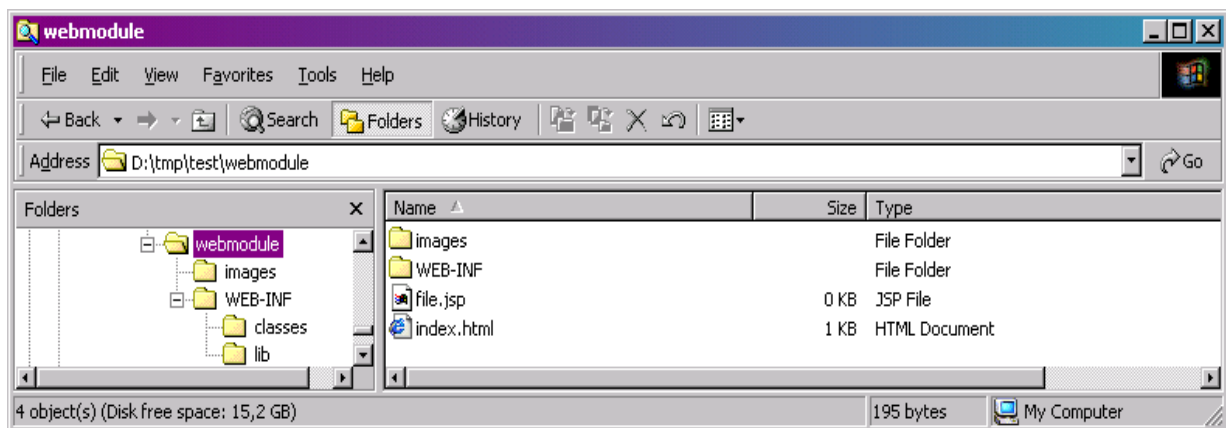
Basicamente a J2EE usa o mesmo mecanismo de publicação e configuração para ambos os componentes.

12.1. Deploy de Módulos Web no JBOSS 3.0

O mecanismo é baseado num arquivo compactado com um conteúdo específico e uma distribuição específica.

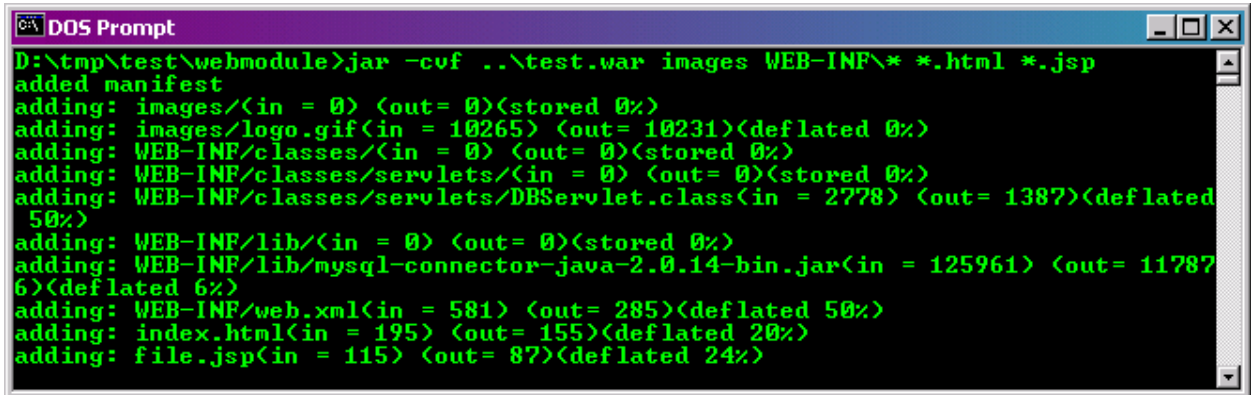
A publicação de componentes web para a J2EE, é feita através de um arquivo **webApp.war** (web archive), que deve ter na sua estrutura o arquivo de configuração (deplo descriptor) para a web, o **web.xml**

Estrutura do módulo web para aplicações J2EE:



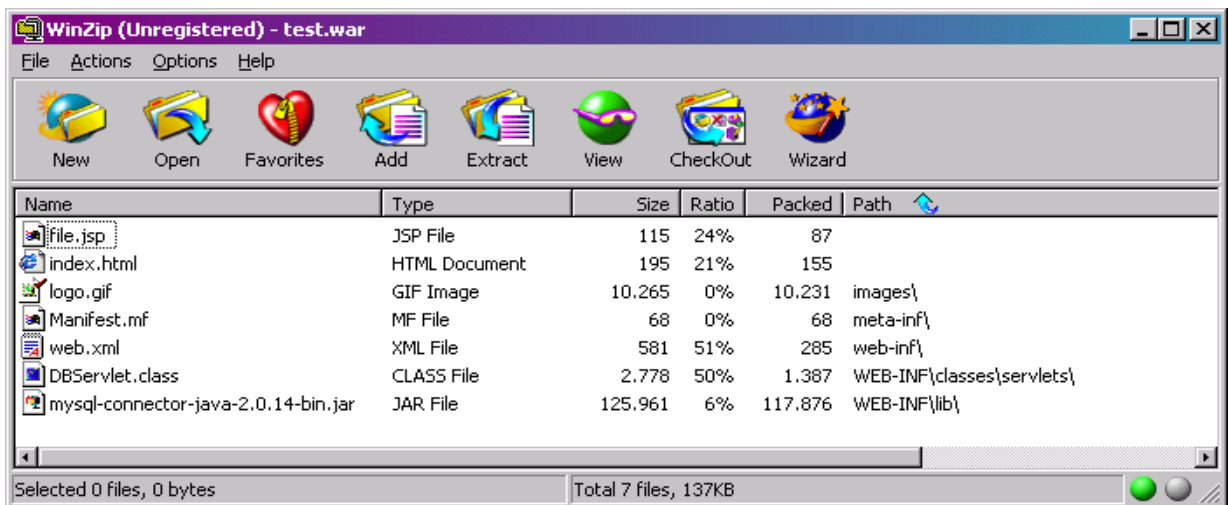
Para se criar o módulo **.war**, basta compactar num arquivo formato **.zip** esta estrutura.

Uma das ferramentas da J2SE, é o **jar**, e podemos usá-lo para compactar a estrutura de arquivos num módulo **.war**.



```
D:\tmp\test\webmodule>jar -cvf ..\test.war images WEB-INF\* *.html *.jsp
added manifest
adding: images/(in = 0) (out= 0)<stored 0%>
adding: images/logo.gif(in = 10265) (out= 10231)<deflated 0%>
adding: WEB-INF/classes/(in = 0) (out= 0)<stored 0%>
adding: WEB-INF/classes/servlets/(in = 0) (out= 0)<stored 0%>
adding: WEB-INF/classes/servlets/DBServlet.class(in = 2778) (out= 1387)<deflated 50%>
adding: WEB-INF/lib/(in = 0) (out= 0)<stored 0%>
adding: WEB-INF/lib/mysql-connector-java-2.0.14-bin.jar(in = 125961) (out= 117876)<deflated 6%>
adding: WEB-INF/web.xml(in = 581) (out= 285)<deflated 50%>
adding: index.html(in = 195) (out= 155)<deflated 20%>
adding: file.jsp(in = 115) (out= 87)<deflated 24%>
```

Estrutura compactada:



O Arquivo **web.xml**, **Web Deploy Descriptor**, indica quais as configurações dos componentes WEB, desde seus nomes, acesso á recursos do container, link de referência no caso de algum componente WEB fazer acesso á algum EJB.

A estrutura de diretórios do módulo WEB é:

- \ conteúdos estáticos como html, imagens, JavaScripts, subdiretórios, etc

- \WEB-INF\web.xml – Deploy Descriptor

- \WEB-INF\classes - usado para armazenar pacotes de classes java compiladas;

- \WEB-INF\lib - usados para armazenar bibliotecas de clases java, em formato **.jar**

ou **.zip**;

- \META-INF\manifest.mf – versão dos arquivos compactados.

Administração e Manutenção do JBOSS Application Server 3.0

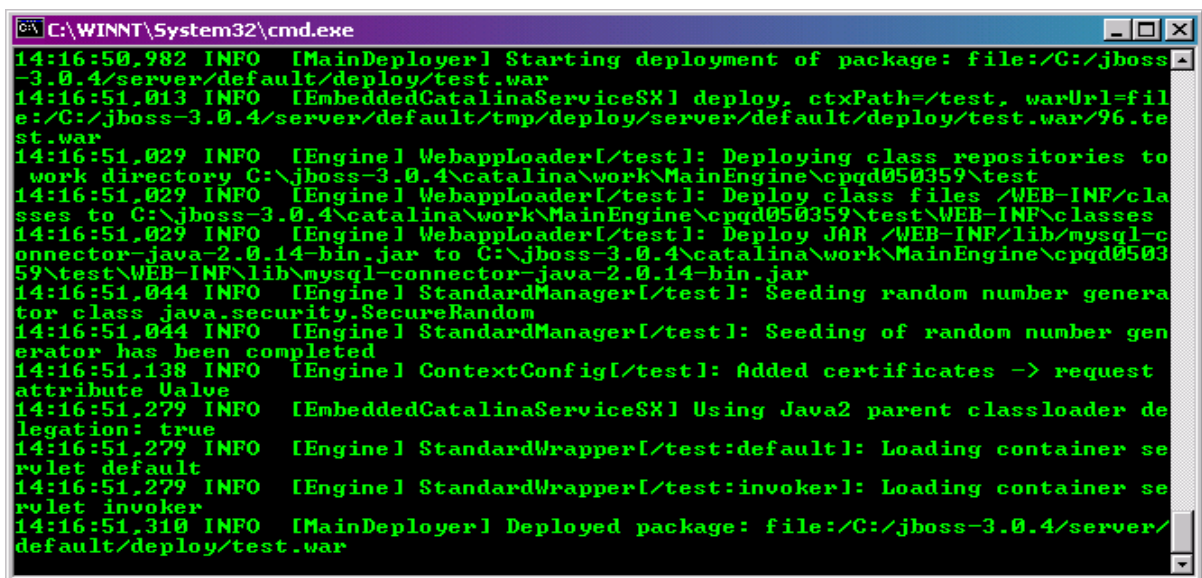
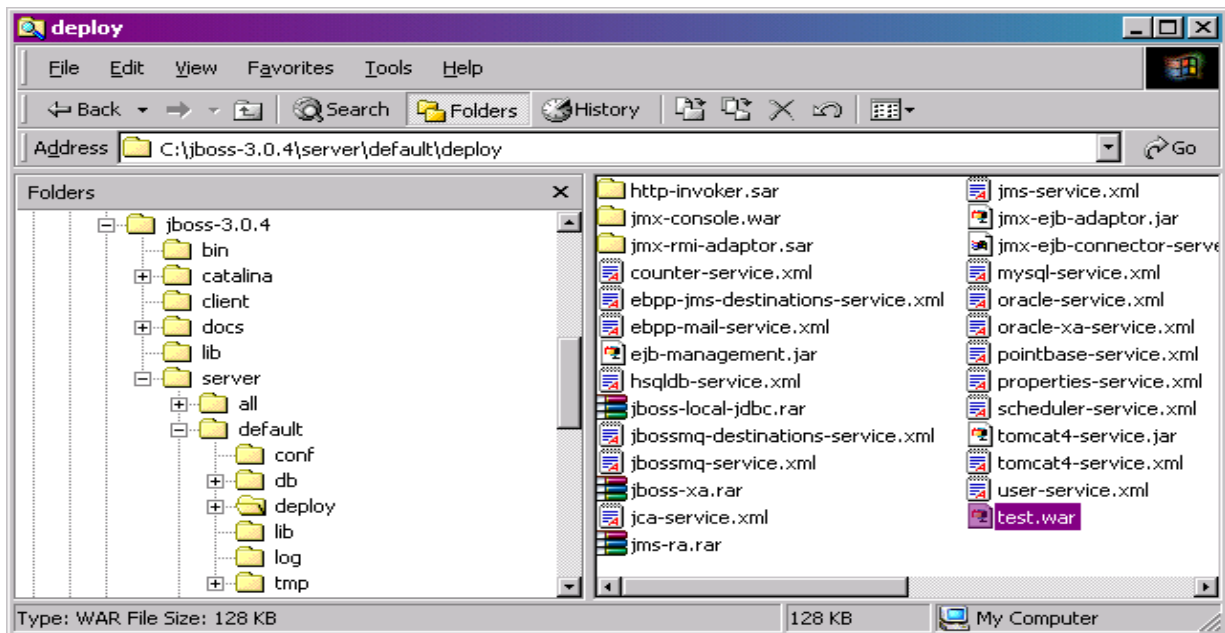
Para se fazer o deploy de módulos WEB no JBOSS 3.0, basta copiar o **.war**, para o diretório de deploy da instância em uso.

Ex: Instância **default**:

%JBOSS_HOME%\server\default\deploy

Copiando o modulo **.war** para o diretório acima, o JBOSS faz a leitura do arquivo, examina o seu conteúdo e faz a publicação da aplicação.

Nota: O nome do contexto web criado é o mesmo nome do módulo **.war**.



Para testar se a aplicação foi publicada com sucesso, basta acessar a aplicação Web.

No navegador web, apontar para:

<http://localhost:8080/test/>

Deve aparecer a página inicial da aplicação.

Examinando o **web.xml**, podemos identificar os componentes presentes na aplicação.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web
Application 2.3//EN"
    'http://java.sun.com/dtd/web-app_2_3.dtd'>

<web-app>

    <display-name>WebTest</display-name>

    <servlet>
        <servlet-name>DBServlet</servlet-name>
        <servlet-class>servlets.DBServlet</servlet-class>
    </servlet>

    <servlet-mapping>
        <servlet-name>DBServlet</servlet-name>
        <url-pattern>/DBServlet</url-pattern>
    </servlet-mapping>

    <session-config>
        <session-timeout>10</session-timeout>
    </session-config>

</web-app>
```

Este arquivo está muito simples, pois este módulo possui apenas um Servlet, e geralmente as aplicações possuem vários Servlets e outros componentes com Filters, Listeners, etc.

Estrutura completa do **web.xml**.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD
Web Application 2.3//EN" "http://java.sun.com/dtd/web-
app_2_3.dtd">
<web-app>
```

<!-- Definição de atributos que são inseridos na inicialização da aplicação, e que podem ser recuperados através do objeto ServletContext. -->

```
    <context-param>
        <param-name>webmaster</param-name>
        <param-value>myaddress@mycompany.com</param-value>
        <description>
            The EMAIL address of the administrator to whom
questions
            and comments about this application should be
addressed.
        </description>
    </context-param>
```

<!-- Definição dos filtros que atuarão a cada chamada a um determinado diretório da aplicação WEB. Os filtros atuam sobre os objetos ServletRequest e ServletResponse -->

```
    <filter>
        <filter-name> FilterName </filter-name>
        <display-name> FilterName </display-name>
        <description> Some Description </description>
        <filter-class>
com.enterprise.application.filter.MyFilter </filter-class>
    </filter>
```

<!-- Definição do diretório da aplicação WEB que quando chamado executará um filtro. -->

```
    <filter-mapping>
        <filter-name> FilterName </filter-name>
        <url-pattern>/*</url-pattern>
    </filter-mapping>
```

<!-- Definição dos Listeners da Aplicação WEB que é executado seguindo a especificação do ServletListener. -->

```
    <listener>
        <listener-class>
com.enterprise.application.listeners.ContextListener
        </listener-class>
    </listener>
```

```

<!-- Definição dos Listeners de Sessão que é executado
segundo a especificação do SessionListener. -->
<listener>
    <listener-class>
com.enterprise.application.listeners.SessionListener
    </listener-class>
</listener>

<!-- Definição dos Servlets presentes na aplicação WEB. -->
<servlet>
    <servlet-name> ServletName </servlet-name>
    <servlet-class> com.enterprise.application.MyServlet
</servlet-class>

    <!-- Definição de atributo que são inseridos na
inicialização, para um Servlet, que poder ser obtido
através do objeto ServletConfig. -->
    <init-param>
        <param-name> parameter </param-name>
        <param-value> parameterValue </param-value>
    </init-param>
</servlet>

<!-- Definição do mapeamento dos Servlets ou JSPs presentes
na aplicação WEB. Este mapeamento representa a URI deste
servlet, ex: http://host/web/myservlet -->
<servlet-mapping>
    <servlet-name> ServletName </servlet-name>
    <url-pattern>/myservlet</url-pattern>
</servlet-mapping>

<!-- Definição do tempo em minutos de Timeout do objeto
HttpSession, que representa a sessão Web de um usuário da
aplicação. -->
<session-config>
    <session-timeout>10</session-timeout>
</session-config>

<!-- Definição nome do arquivo padrão para diretórios na
aplicação. -->
<welcome-file-list>
    <welcome-file>index.html</welcome-file>
</welcome-file-list>

```



```

<!-- Mapeamento do nome lógico/físico que os JSPs usarão
para identificar uma TagLib. -->
<taglib>
  <taglib-uri>/taglibrary.tld</taglib-uri>
  <taglib-location>/WEB-INF/taglibrary.tld</taglib-
location>
</taglib>

<!-- Definição para permitir o acesso á um determinado
Recurso do Container, para que o container possa propagar o
Contexto de execução das Requisições. -->
<resource-ref>
  <res-ref-name>jdbc/DataSource</res-ref-name>
  <res-type>javax.sql.DataSource</res-type>
  <res-auth>Container</res-auth>
</resource-ref>

<!-- Definição para habilitar o Security Framework da J2EE.
Este framework é "Role-Based", onde uma Determinada "Role"
possuirá usuários (JAAS) -->
<security-constraint>
  <display-name>Example Security Constraint</display-
name>
  <web-resource-collection>
    <web-resource-name>Protected Area</web-resource-
name>
    <!-- Define the context-relative URL(s) to be
protected -->
    <url-pattern>/security/*</url-pattern>
    <!-- If you list http methods, only those methods are
protected -->
    <!-- E uma vez autenticado, este usuário poderá acessar
estes recursos de acordo com as seguintes restrições. -->
    <http-method>GET</http-method>
    <http-method>POST</http-method>
  </web-resource-collection>
  <auth-constraint>
    <!-- Anyone with one of the listed roles may
access this area -->
    <role-name>HRUsers</role-name>
    <role-name>Admin</role-name>
  </auth-constraint>
</security-constraint>

```

```

<!-- Método de autenticação que será usado pela aplicação,
após a autenticação, o Container passara á fase de
autorização descrita nas SecurityConstraints. Os métodos
de autenticação podem ser BASIC, FORM, CERT-SIGN -->
    <login-config>
        <auth-method>FORM</auth-method>
        <realm-name>Example Form-Based Authentication
Area</realm-name>
        <form-login-config>
            <form-login-page>/security/login.jsp</form-login-
page>
            <form-error-page>/security/error.jsp</form-error-
page>
        </form-login-config>
    </login-config>

<!-- Definição para o registro de uma variável no serviço
de nomes (JNDI) do servidor J2EE. -->
    <env-entry>
        <env-entry-name> jndi_envirominent_name </env-entry-
name>
        <env-entry-value> jndi_envirominent_value </env-entry-
value>
        <env-entry-type> jndi_envirominent_type </env-entry-
type>
    </env-entry>

<!-- Definição para permitir o acesso á um determinado EJB,
para que o container possa propagar o Contexto de execução
das Requisições. -->
    <ejb-ref>
        <description>Example EJB Reference</description>
        <ejb-ref-name>ejb/ObjectEJB</ejb-ref-name>
        <ejb-ref-type>Session</ejb-ref-type>

    <home>com.enterprise.application.ejb.ObjectHome</home>

    <remote>com.enterprise.application.ejb.Object</remote>
    </ejb-ref>
</web-app>

```

Descobrimos então que cada Aplicação terá arquivos que têm a mesma estrutura, mas com valores completamente diferentes.

Estrutura básica do taglibrary.tld

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE taglib
    PUBLIC "-//Sun Microsystems, Inc.//DTD JSP Tag
    Library 1.2//EN"
    "http://java.sun.com/dtd/web-
    jsptaglibrary_1_2.dtd">
<taglib>
    <tlib-version>1.0</tlib-version>
    <jsp-version>1.2</jsp-version>

<!-- Definição de um "short-name" que poderá ser usado para
chamar a tag. -->
    <short-name> tag </short-name>
    <description> Some Tag.</description>
    <tag>

<!-- Definição de um "name" que poderá ser usado para
chamar a tag, e qual a classe que a definiu, e como será
executado o conteúdo interno da tag. -->
    <name> printTable </name>
    <tag-class>
com.enterprise.application.tag.PrintTable</tag-class>
    <body-content>JSP</body-content>
    <description> Generates a HTML Table</description>
    </tag>
</taglib>
```

**Descobrimos também, que cada TagLib terá seu
DeployDescriptor de configuração.**

12.2. Deploy de Módulos EJB no JBOSS 3.0

Os EJBs (Enterprise JavaBeans) são componentes Java que seguem a especificação J2EE e tem como principal responsabilidade fornecer um ambiente amigável de execução de processos de negócios, e ainda fornecer serviços sem necessidade de implementação para transação, segurança, persistência e gerência de recursos.

Além disso, eles devem controlar a concorrência das requisições e permitir acesso distribuído.

Os EJBs da J2EE 1.3 seguem as seguintes especificações: EJB 2.0

Session Bean Stateless – Somente Transacionais, e não mantem estado entre as chamadas de métodos;

Session Bean Stateful – Transacionais, e mantem estado entre as chamadas de métodos.

Entity Bean – Usado para implementar o mapeamento Objeto-Relacional e suportar os serviços de persistência.

BMP – Bean Managed Persistence – o código que implementa a persistência deve ser construído podendo ser feito em arquivo, bancos de dados SQL, etc.

CMP – Container Manager Persistence – o EJB Container do JBOSS se encarregar de definir “onde” e configurar “como” a persistência das entidades de dados será feita.

Message Driven Bean – Usado para consumir mensagens do JMS na forma assíncrona. Trás para a J2EE a capacidade de executar processos usando o conceito de filas. A implementação das filas pode ser Queuer (Peer-To-Peer) ou Topic (Publish – Subscribe).

Os SessionBeans e os EntityBeans, tem particularidades e diferenças na sua construção, entretanto eles seguem um modelo.

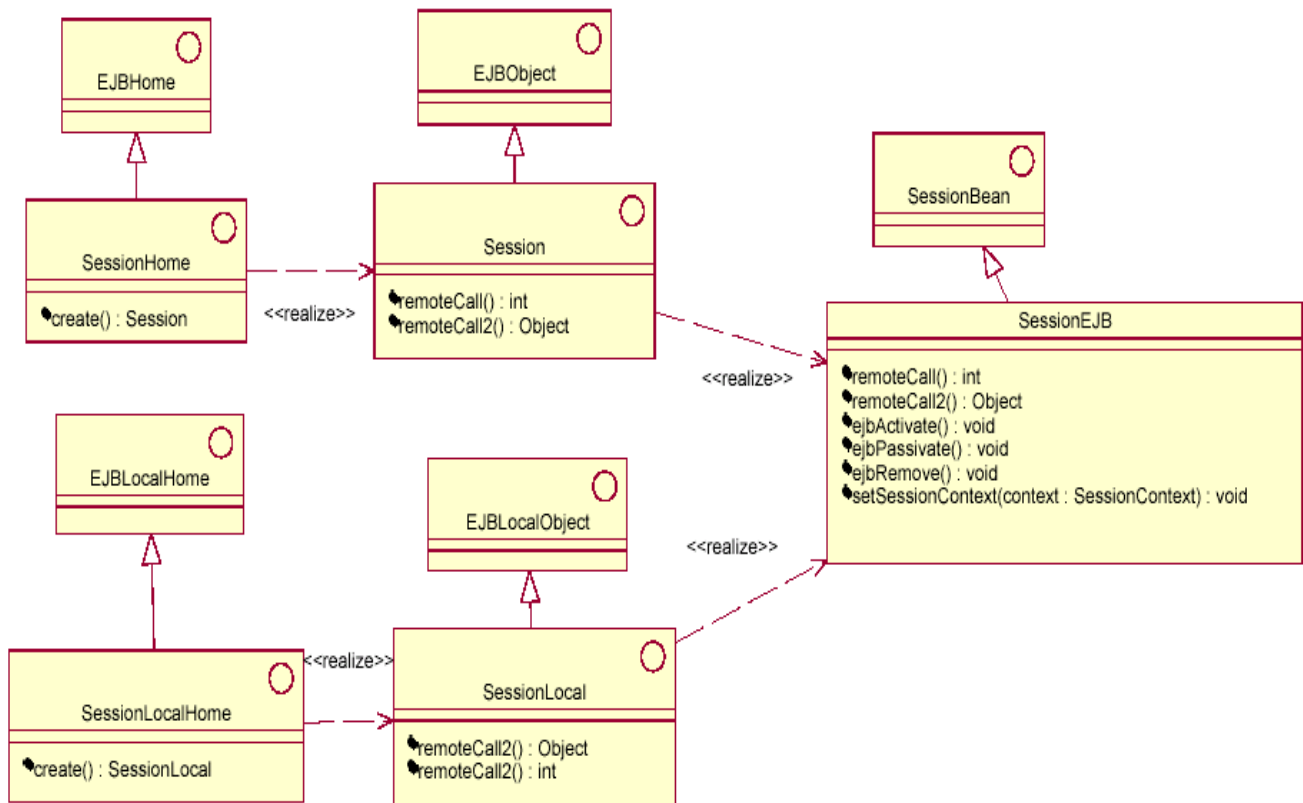
Esses dois tipos foram construídos para terem 3 visões. Uma para os Clientes da aplicação, uma para a Aplicação e outra para o Container. Cada visão se refere á um conjunto de métodos que deve ser implementado no EJB.

Os EJBs são extremamente usados nas implementações de “business” de alta-performance e escalabilidade baseados nos modelos da J2EE. E em 90% dos casos também são usados como estratégia de implementação de Enterprise Design Patterns, ou atualmente chamados de J2EE Design Patterns.

As partes físicas dos SessionBeans são:

- 1 Classe “concreta” que implementa uma interface javax.ejb.SessionBean; e
- 1 Interface Remote que estende a javax.ejb.EJBObject e 1 Interface Home que estende a javax.ejb.EJBHome; e/ou
- 1 Interface Local Remote que estende a javax.ejb.EJBLocalObject e 1 Interface Local Home que estende a javax.ejb.EJBLocal Home;

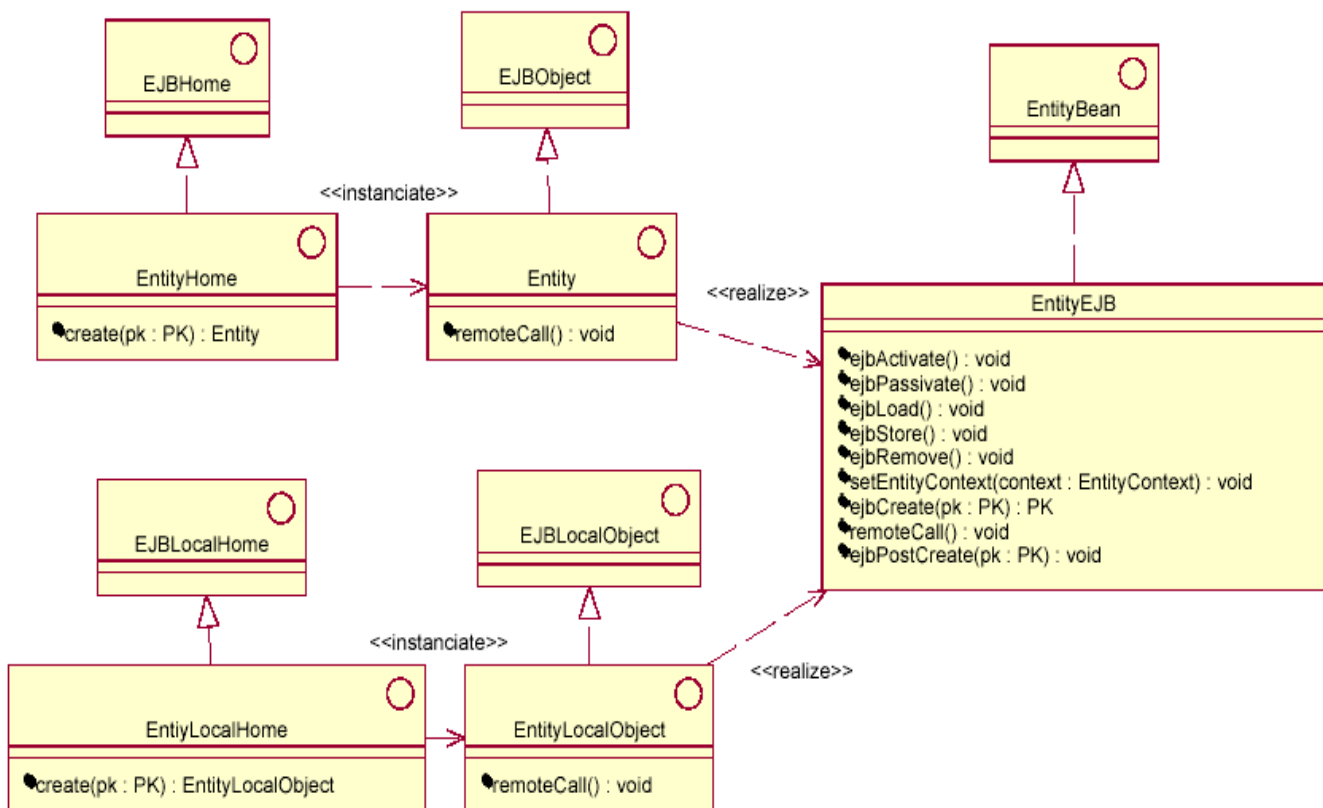
Modelo do Session Bean:



As partes físicas dos EntityBeans são:

- d. 1 Classe “concreta” que implementa uma interface javax.ejb.EntityBean; e
- e. 1 Interface Remote que estende a javax.ejb.EJBObject e 1 Interface Home que estende a javax.ejb.EJBHome; e/ou
- f. 1 Interface Local Remote que estende a javax.ejb.EJBLocalObject e 1 Interface Local Home que estende a javax.ejb.EJBLocal Home;

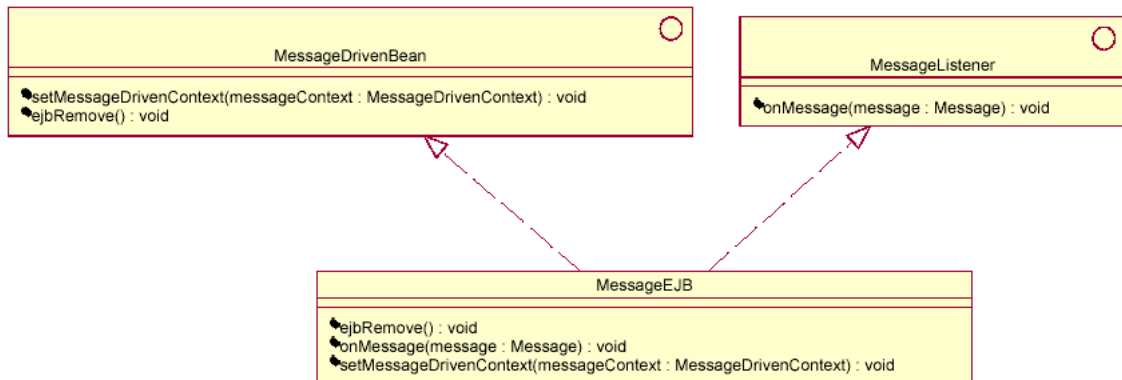
Modelo do Entity Bean:



As partes físicas dos MessageDrivenBean são:

- g. 1 Classe “concreta” que implementa uma interface `javax.ejb.MessageDrivenBean`; e
- h. estar associado a um Queue ou Topic do JMS do JBOSS;

Modelo do Message Driven Bean:

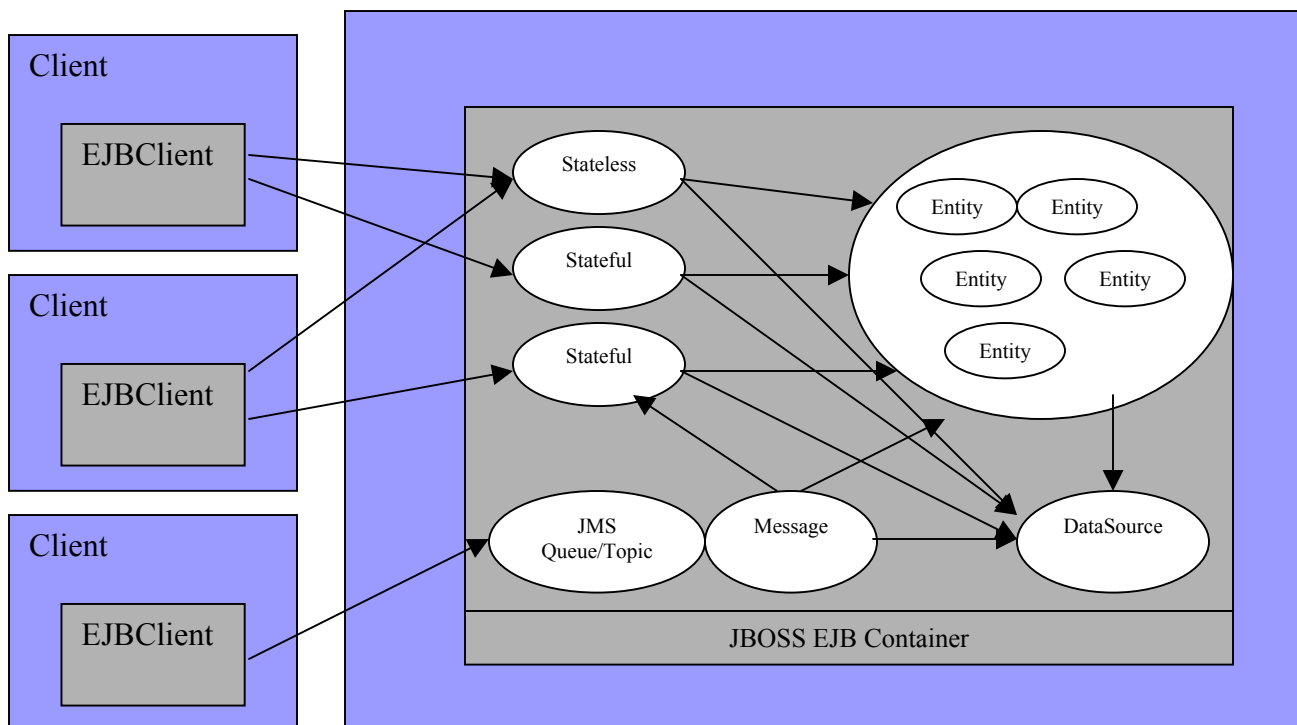


A classe concreta do EJB, deve sempre implementar uma Interface específica que deriva da `javax.ejb.EnterpriseBean`, e será isso que determinará seu tipo.

Visões:

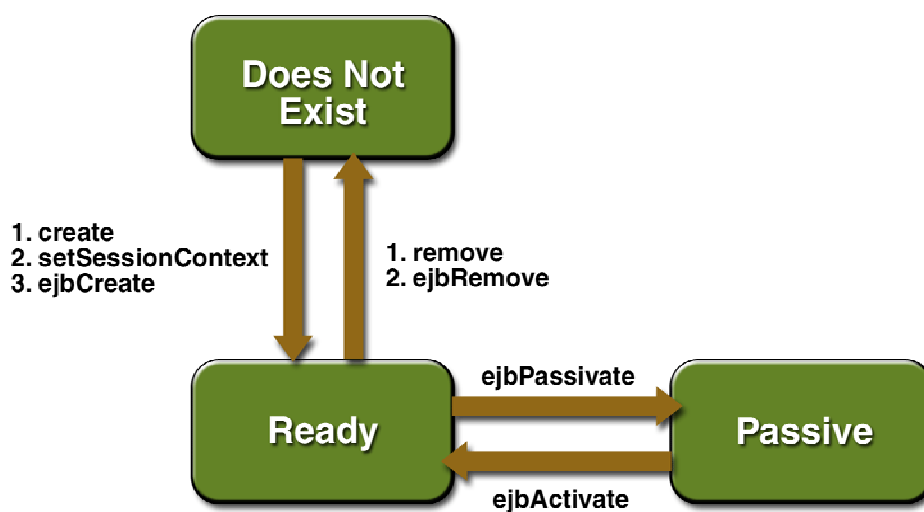
- i. todos os métodos que começam com `ejb....` são exclusivamente chamados pelo Container. (Visão do Container)
- ii. todos os métodos que setam o Contexto (`setEntityContext`, `setSessionContext` e `setMessageDrivenContext`), são exclusivamente chamados pelo Container; (Visão do Container)
- iii. os métodos presentes nas interfaces “remotas” podem ser chamados por qualquer cliente do EJB, interno ou externo; (Visão de Cliente Externo ou Interno)
- iv. os métodos presentes nas interfaces “locais” só podem ser chamados por um cliente do EJB, interno ao mesmo Container; (Visão de Cliente Interno Somente)

O JBOSS Application Server 3.0, possui um EJB Container 2.0, que suporta os três tipos de EJBs presentes na J2EE 1.3,



Devemos também entender o ciclo de vida do EJBs. E cada um deles, segue um comportamento diferente.

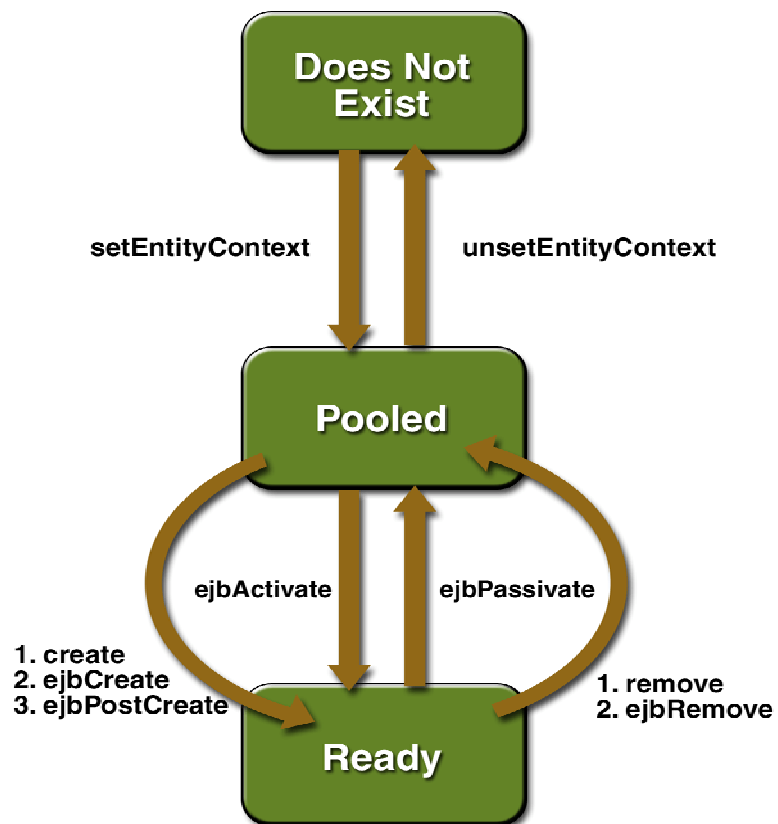
Ciclo de vida dos Stateful Session Beans:



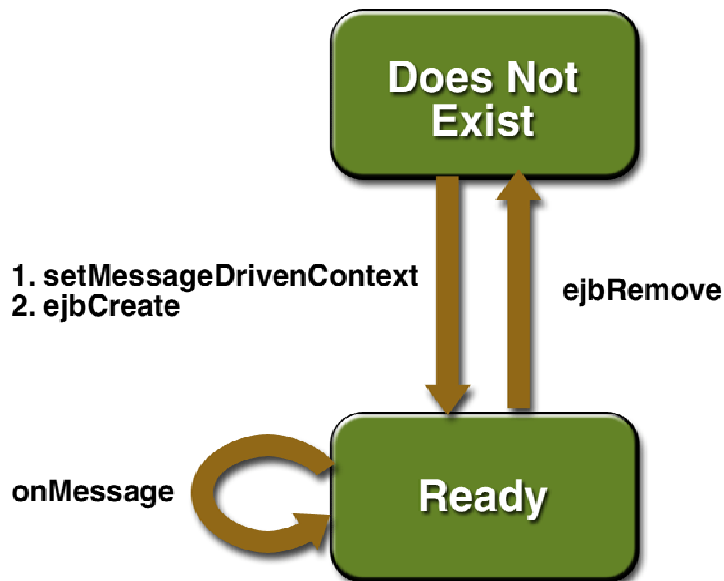
Ciclo de vida dos Stateless Session Beans:



Ciclo de vida dos Entity Beans:



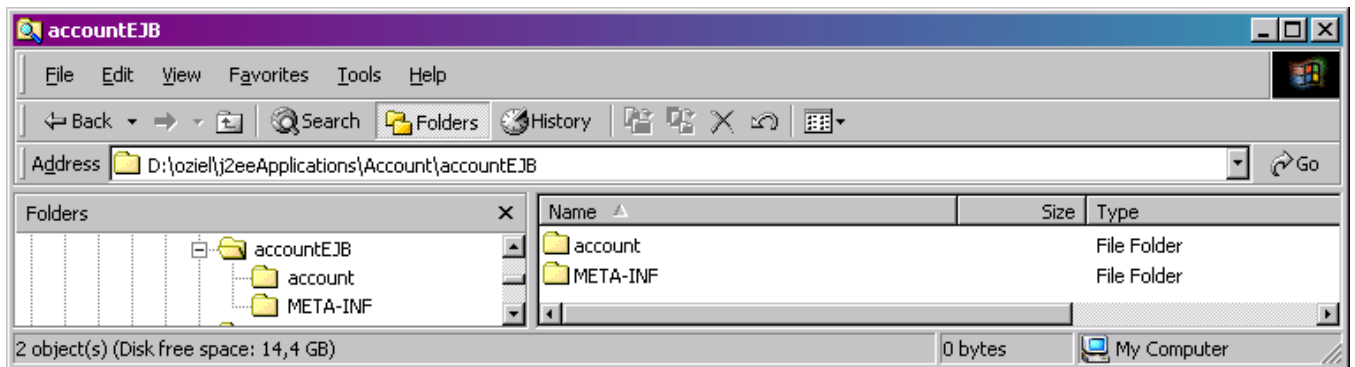
Ciclo de vida dos Message Driven Beans:



Referência completa sobre os EJBs:

http://java.sun.com/j2ee/tutorial/1_3-fcs/doc/EJBConcepts9.html

O Deploy dos EJBs se dá criando um **.jar**, numa estrutura padronizada que segue:



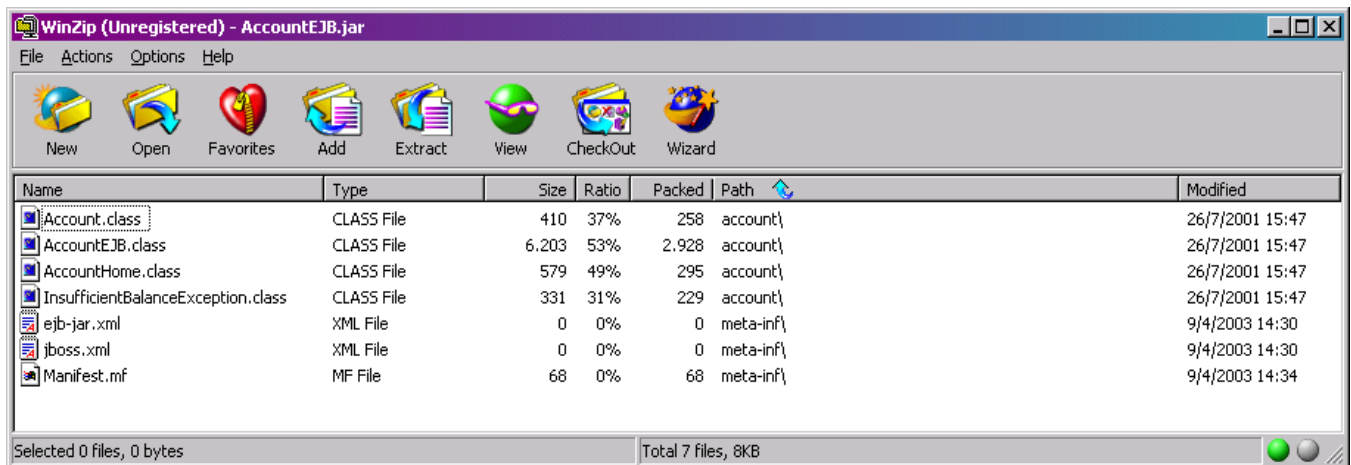
É dentro do diretório **META-INF** onde fica o **Deploy Descriptor** para o Módulo EJB. E no caso do JBOSS 3.0, pode aparecer também com o **ejb-jar.xml** um arquivo **jboss.xml**, que serve para configurações proprietárias do JBOSS que não fazem parte da especificação Standard da J2EE 1.3, como configurações do Cluster, extensão da implementação do JAAS, configurações de persistência, gerência de recursos, etc.

Construindo o Pacote do EJB.jar:

```
DOS Prompt

D:\oziel\j2eeApplications\Account\accountEJB>jar cvf ..\AccountEJB.jar *
added manifest
adding: account/(in = 0) (out= 0)(stored 0%)
adding: account/Account.class(in = 410) (out= 258)(deflated 37%)
adding: account/AccountEJB.class(in = 6203) (out= 2928)(deflated 52%)
adding: account/AccountHome.class(in = 579) (out= 295)(deflated 49%)
adding: account/InsufficientBalanceException.class(in = 331) (out= 229)(deflated
30%)
ignoring entry META-INF/
adding: META-INF/ejb-jar.xml(in = 0) (out= 0)(stored 0%)
adding: META-INF/jboss.xml(in = 0) (out= 0)(stored 0%)
```

Estrutura descompactada:

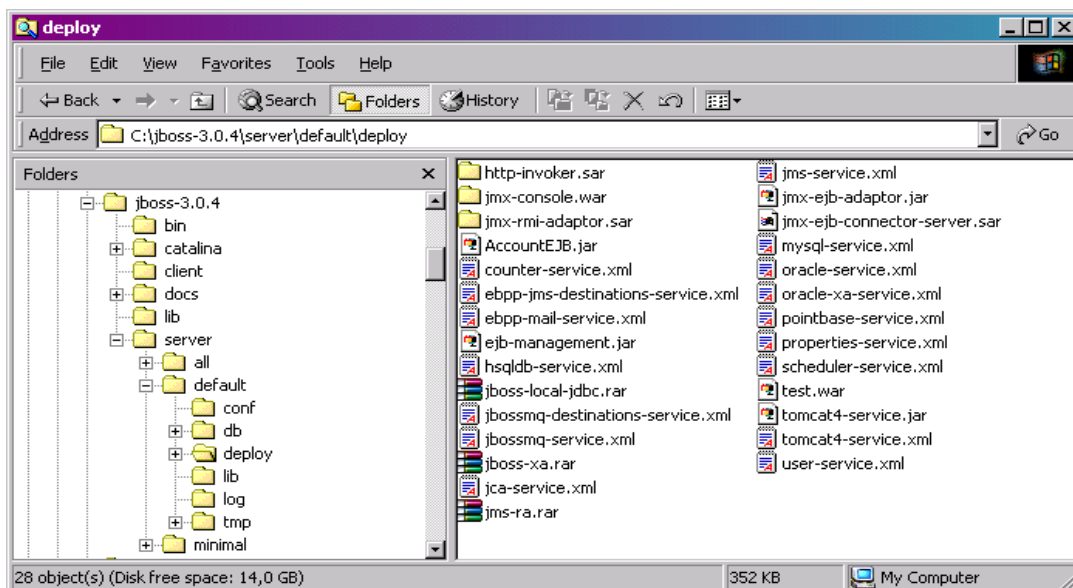


WinZip (Unregistered) - AccountEJB.jar

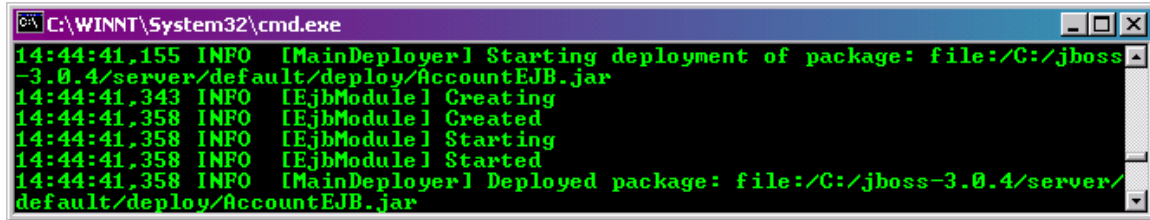
Name	Type	Size	Ratio	Packed	Path	Modified
Account.class	CLASS File	410	37%	258	account\	26/7/2001 15:47
AccountEJB.class	CLASS File	6,203	53%	2,928	account\	26/7/2001 15:47
AccountHome.class	CLASS File	579	49%	295	account\	26/7/2001 15:47
InsufficientBalanceException.class	CLASS File	331	31%	229	account\	26/7/2001 15:47
ejb-jar.xml	XML File	0	0%	0	meta-inf\	9/4/2003 14:30
jboss.xml	XML File	0	0%	0	meta-inf\	9/4/2003 14:30
Manifest.mf	MF File	68	0%	68	meta-inf\	9/4/2003 14:34

Selected 0 files, 0 bytes Total 7 files, 8KB

Fazendo Deploy. Basta copiar no diretório de deploy do JBOSS 3.0



Verificando o Deploy:



```

C:\WINNT\System32\cmd.exe
14:44:41,155 INFO [MainDeployer] Starting deployment of package: file:/C:/jboss-3.0.4/server/default/deploy/AccountEJB.jar
14:44:41,343 INFO [EjbModule] Creating
14:44:41,358 INFO [EjbModule] Created
14:44:41,358 INFO [EjbModule] Starting
14:44:41,358 INFO [EjbModule] Started
14:44:41,358 INFO [MainDeployer] Deployed package: file:/C:/jboss-3.0.4/server/default/deploy/AccountEJB.jar
  
```

Para testar, é necessário executar um cliente externo (Java Client) ou um cliente interno Web (Servlet ou JSP).

Entendo o Deploy Descriptor dos EJBs:

Estrutura básica do ejb-jar.xml

```

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE ejb-jar PUBLIC "-//Sun Microsystems, Inc.//DTD Enterprise JavaBeans
2.0//EN" "http://java.sun.com/dtd/ejb-jar_2_0.dtd">
<ejb-jar>
  <enterprise-beans>
    <session>
      <display-name>ObjectEJB</display-name>
      <ejb-name>ObjectEJB</ejb-name>
      <home>com.enterprise.application.ejb.ObjectHome</home>
      <remote>com.enterprise.application.ejb.Object</remote>
      <ejb-class>com.enterprise.application.ejb.ObjectEJB</ejb-class>
      <session-type>Stateless</session-type>
      <transaction-type>Container</transaction-type>
      <env-entry>
        <env-entry-name> jndi_environminent_name </env-entry-name>
        <env-entry-value> jndi_environminent_value </env-entry-
value>
        <env-entry-type> jndi_environminent_type </env-entry-type>
      </env-entry>
      <resource-ref>
        <res-ref-name>jdbc/Oracle</res-ref-name>
        <res-type>javax.sql.DataSource</res-type>
        <res-auth>Container</res-auth>
      </resource-ref>
    </session>
    <message-driven>
      <display-name>MessageEJB</display-name>
      <ejb-name>MessageEJB</ejb-name>
      <ejb-class>com.enterprise.application.ejb.MessageEJB</ejb-class>
      <transaction-type>Bean</transaction-type>
      <acknowledge-mode>Auto-acknowledge</acknowledge-mode>
      <message-driven-destination>
        <destination-type>javax.jms.Queue</destination-type>
      </message-driven-destination>
      <env-entry>
        <env-entry-name> jndi_environminent_name </env-entry-name>
        <env-entry-value> jndi_environminent_value </env-entry-
value>
        <env-entry-type> jndi_environminent_type </env-entry-type>
      </env-entry>
  
```

```
<resource-ref>
  <res-ref-name>jdbc/Oracle</res-ref-name>
  <res-type>javax.sql.DataSource</res-type>
  <res-auth>Container</res-auth>
</resource-ref>
</message-driven>
</enterprise-beans>

<assembly-descriptor>
  <security-role>
    <description>The single application role</description>
    <role-name>HRUsers</role-name>
  </security-role>
  <method-permission>
    <description>The employee and temp-employee roles may
      access any method of the EmployeeService bean
    </description>
    <role-name>HRUsers</role-name>
    <method>
      <ejb-name>ObjectEJB</ejb-name>
      <method-name>*</method-name>
    </method>
    <method>
      <ejb-name>MessageEJB</ejb-name>
      <method-name>*</method-name>
    </method>
  </method-permission>
  <container-transaction>
    <method>
      <ejb-name>ObjectEJB</ejb-name>
      <method-name>*</method-name>
    </method>
    <trans-attribute>Required</trans-attribute>
  </container-transaction>
</assembly-descriptor>
</ejb-jar>
```

Estrutura básica do jboss.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE jboss PUBLIC "-//JBoss//DTD JBOSS//EN"
"http://www.jboss.org/j2ee/dtd/jboss.dtd">

<jboss>

  <enterprise-beans>
  </enterprise-beans>

  <resource-managers>
  </resource-managers>

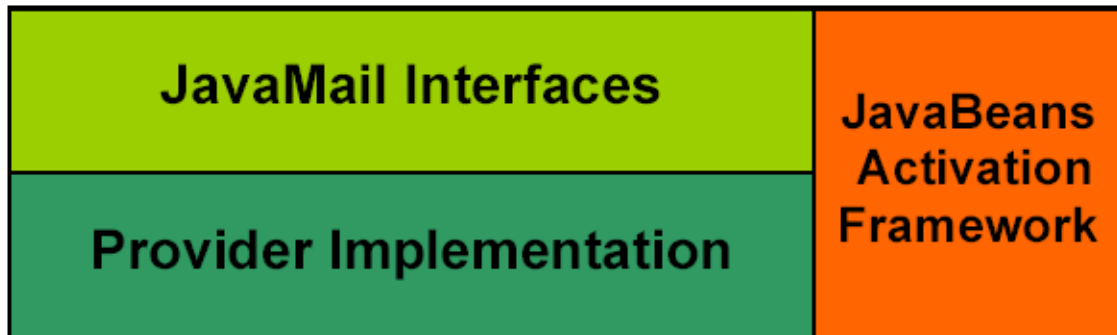
</jboss>
```

Empacotando Aplicações Exemplo e fazendo Deploy dos EJBs.

13. Configurando o JavaMail no JBOSS 3.0

O JavaMail é uma API padrão da J2EE 1.3 também, ou seja, o JBOSS 3.0 implementa um conjunto de componentes configuráveis que permitem a criação de objetos na JNDI que representam os recursos de um MailServer.

O JavaMail usa o JAF (JavaBeans Activation Framework) para garantir o bom uso dos recursos de memória, conexões, etc.



Os principais componentes do JavaMail são:

- javax.mail.Service: Abstração dos serviços de email;
- javax.mail.Session: Representa uma sessão de usuário no mail server;

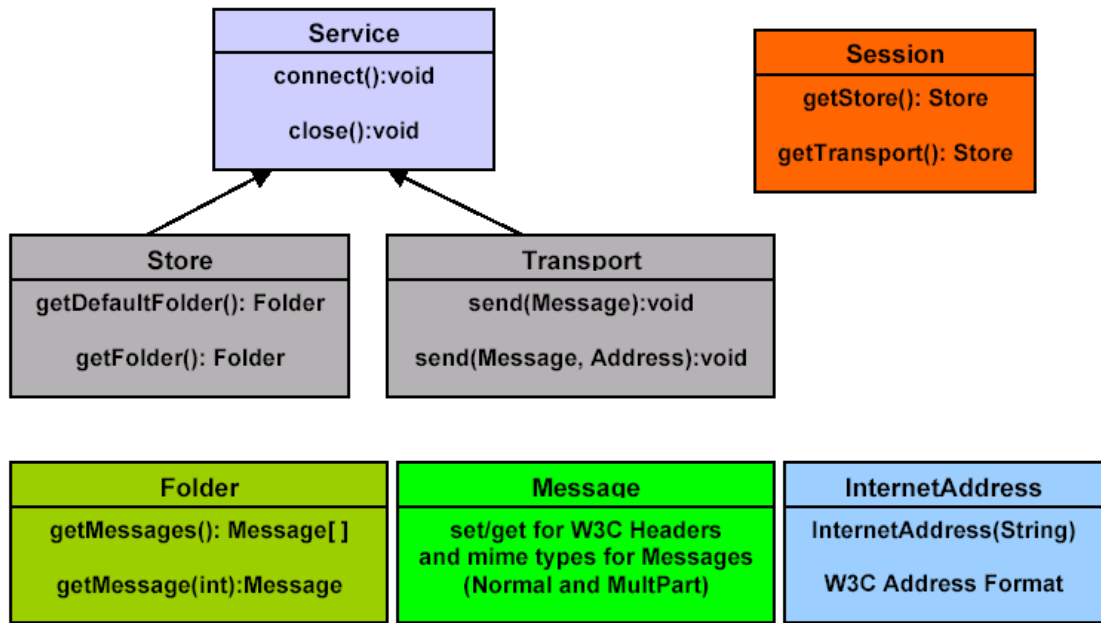
A partir de uma Session, é possível se obter dois objetos:

- javax.mail.Transport: Representa a conexão SMTP ou IMAP para envio de mensagens;
- javax.mail.Store: Representa a conexão POP3 ou IMAP para leitura de mensagens e caixas postais.

O SMTP é o serviço responsável pelo envio das mensagens, o POP3 é o serviço responsável pelo armazenamento das mensagens e o IMAP é uma evolução para um modelo de sincronismo e aglutinação dos serviços de envio e armazenamento.

A JavaSoft e outros fabricantes, criaram as suas implementações (Provider) destes serviços, ou seja, criaram as classes que implementam a interface padrão das especificações.

Arquitetura do JavaMail:



Exemplo de envio de email usando o recurso do JavaMail dentro do JBOSS:

```

private Session session;
private Transport smtp;
...
public void init() {
    try {
        InitialContext jndi = new InitialContext();
        session = jndi.lookup("java:/Mail");
        smtp = session.getTransport();
        jndi.close();
    } catch (Exception e) {
        e.printStackTrace();
    }
}

public void send(String from, String to, String subject, String
text) throws Exception {
    InternetAddress toAddress = new InternetAddress(to);
    InternetAddress fromAddress = new InternetAddress(from);
    MimeMessage message = new MimeMessage( session );
    message.setFrom( fromAddress );
    message.setSubject( subject );
    message.setText( text );
    message.setSentDate( new Date() );
    message.addRecipients( Message.RecipientType.TO, to);
    smtp.send( message );
    smtp.close();
}

```

Configurando o serviço do JavaMail dentro do JBOSS.

Basta criar um arquivo **xxxx-service.xml**, e colocá-lo dentro do diretório:
 %JBOSS_HOME%\server\<instance>\deploy.

mail-service.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE server>
<server>
  <classpath codebase="lib"
    archives="mail.jar, activation.jar, mail-plugin.jar"/>

  <mbean code="org.jboss.mail.MailService"
    name="jboss:service=Mail">
    <attribute name="JNDIName">java:/Mail</attribute>
    <attribute name="User">userName</attribute>
    <attribute name="Password">password</attribute>
    <attribute name="Configuration">
      <configuration>
        <!-- Change to your mail server protocol -->
        <property name="mail.store.protocol" value="pop3"/>
        <property name="mail.transport.protocol" value="smtp"/>

        <!-- Change to the user who will receive mail -->
        <property name="mail.user" value="userName"/>
        <!-- Change to the mail server -->
        <property name="mail.pop3.host"
          value="mailsrv1.aquarius.cpqd.com.br"/>
        <!-- Change to the SMTP gateway server -->
        <property name="mail.smtp.host"
          value="mailsrv1.aquarius.cpqd.com.br"/>
        <!-- Change to the address mail will be from -->
        <property name="mail.from" value="userName@cpqd.com.br"/>
        <!-- Enable debugging output from the javamail classes -->
        <property name="mail.debug" value="false"/>
      </configuration>
    </attribute>
  </mbean>
</server>
```

Deploy de uma aplicação Web que envia emails usando o JavaMail.

14. Configurando o JMS no JBOSS 3.0

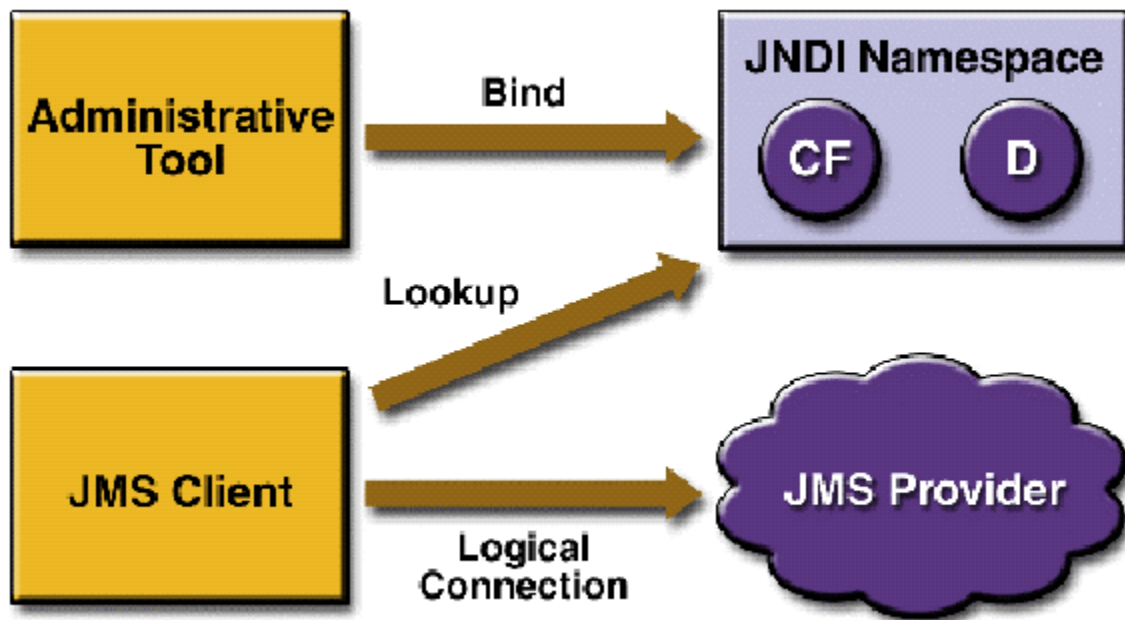
O JMS (JavaMessage Services) também uma API padrão da J2EE 1.3, e foi construída com o intuito de permitir processos assíncronos dentro das aplicações J2EE.

Pacotes da API Java e suas funcionalidades:

`javax.jms.*`; API que contém as interfaces padronizadas pela J2EE.

Arquitetura de serviços da API JMS:

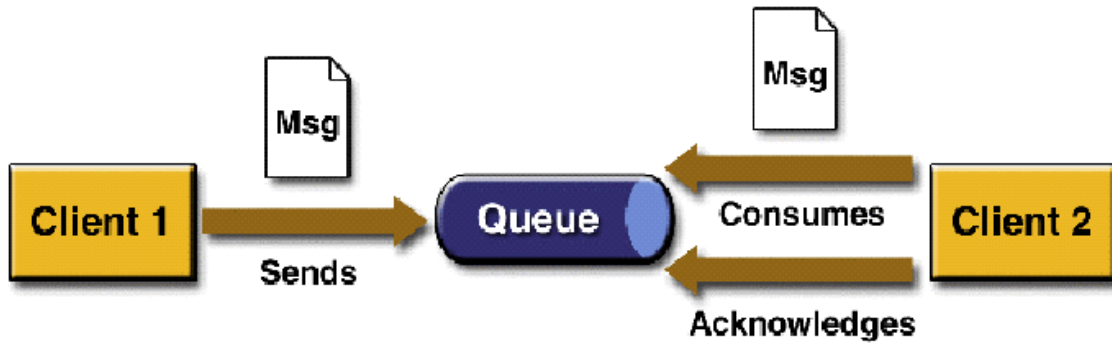
Este serviço já está integrado com o Contexto das aplicações J2EE e pode utilizar todas as suas tecnologias como Segurança, acesso a EJBs, acesso ao DataSource (JDBC), utilização do monitor transacional, o serviço JMS quando utilizado implementa persistência das mensagens trocadas pois há necessidade de garantia de entrega.



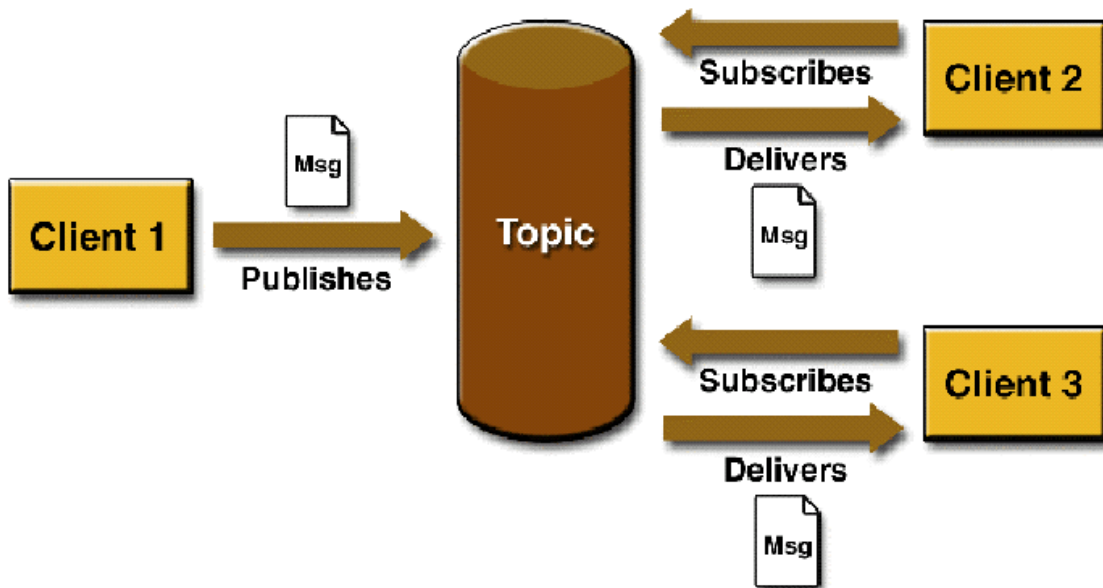
Uma vez configurado, a sua utilização é bem simples. E ainda podemos usar o JMS integrado aos EJB, usando os Message Driven Beans.

Modelos de implementação:

O serviço de **“Queue”** é usado para trocar mensagens usando PTP (Point-to-Point), ou seja, alguém gera uma mensagem que será “lida e entendida” somente uma vez, e somente por um único destino, e enquanto não for “entendida” lida fica armazenada de acordo com seu TTL (“Time To Live”);



O serviço de **“Topic”** é usado para trocar mensagens usando o conceito de Publisher (que coloca uma ou mais mensagens na fila), e o Subscriber (que recebe um evento para lê a mensagem da fila), e enquanto não for “entendida” lida fica armazenada de acordo com seu TTL (“Time To Live”);



Existe um arquivo **jbossmq-sevice.xml** dentro do diretório de deploy, que configura o “**Provider**”, “**Resources**” e “**Connections**” do JMS para o JBOSS, incluindo a forma de Persistência, etc.

Um Topic ou Queue pode ser persistido em Banco de Dados, FileSystem, ou outro repositório.

Arquivo jbossmq-service.xml (configuração Standard JBOSS 3.0)

```
<?xml version="1.0" encoding="UTF-8"?>
  <!--
    | InvocationLayers are the different transport methods that can
    | be used to access the server.
  -->
  <mbean code="org.jboss.mq.il.jvm.JVMServerILService"
        name="jboss.mq:service=InvocationLayer,type=JVM">
    <depends optional-attribute-
name="Invoker">jboss.mq:service=Invoker</depends>
    <attribute
name="ConnectionFactoryJNDIRef">java:/ConnectionFactory</attribute>
    <attribute
name="XAConnectionFactoryJNDIRef">java:/XAConnectionFactory</attribute>
    <attribute name="PingPeriod">0</attribute>
  </mbean>

  <mbean code="org.jboss.mq.il.rmi.RMIServerILService"
        name="jboss.mq:service=InvocationLayer,type=RMI">
    <depends optional-attribute-
name="Invoker">jboss.mq:service=Invoker</depends>
    <attribute
name="ConnectionFactoryJNDIRef">RMIConnectionFactory</attribute>
    <attribute
name="XAConnectionFactoryJNDIRef">RMIXAConnectionFactory</attribute>
    <attribute name="PingPeriod">60000</attribute>
  </mbean>

  <mbean code="org.jboss.mq.il.oil.OILServerILService"
        name="jboss.mq:service=InvocationLayer,type=OIL">
    <depends optional-attribute-
name="Invoker">jboss.mq:service=Invoker</depends>
    <attribute
name="ConnectionFactoryJNDIRef">ConnectionFactory</attribute>
    <attribute
name="XAConnectionFactoryJNDIRef">XAConnectionFactory</attribute>
    <attribute name="ServerBindPort">8090</attribute>
    <attribute name="PingPeriod">60000</attribute>
    <attribute name="EnableTcpNoDelay">true</attribute>
  </mbean>
```

```

    <mbean code="org.jboss.mq.il.uil.UILServerILService"
        name="jboss.mq:service=InvocationLayer,type=UIL">
        <depends optional-attribute-
name="Invoker">jboss.mq:service=Invoker</depends>
        <attribute
name="ConnectionFactoryJNDIRef">UILConnectionFactory</attribute>
        <attribute
name="XAConnectionFactoryJNDIRef">UILXAConnectionFactory</attribute>
        <attribute name="ServerBindPort">8091</attribute>
        <attribute name="PingPeriod">60000</attribute>
        <attribute name="EnableTcpNoDelay">true</attribute>
    </mbean>
    <!-- JBossMQ Interceptor chain configuration -->
    <mbean code="org.jboss.mq.server.jmx.Invoker"
name="jboss.mq:service=Invoker">
        <depends optional-attribute-
name="NextInterceptor">jboss.mq:service=TracingInterceptor</depends>
    </mbean>
    <mbean code="org.jboss.mq.server.jmx.InterceptorLoader"
name="jboss.mq:service=TracingInterceptor">
        <attribute
name="InterceptorClass">org.jboss.mq.server.TracingInterceptor</attribu
te>
        <depends optional-attribute-
name="NextInterceptor">jboss.mq:service=SecurityManager</depends>
    </mbean>
    <mbean code="org.jboss.mq.security.SecurityManager"
name="jboss.mq:service=SecurityManager">
        <depends optional-attribute-
name="NextInterceptor">jboss.mq:service=DestinationManager</depends>
    </mbean>
    <mbean code="org.jboss.mq.server.jmx.DestinationManager"
name="jboss.mq:service=DestinationManager">
        <depends optional-attribute-
name="PersistenceManager">jboss.mq:service=PersistenceManager</depends>
        <depends optional-attribute-
name="StateManager">jboss.mq:service=StateManager</depends>
    </mbean>

```

```

<!--
    | The MessageCache decides where to put JBossMQ message that
    | are sitting around waiting to be consumed by a client.
    | The memory marks are in Megabytes.  Once the JVM memory usage
hits
    | the high memory mark, the old messages in the cache will start
getting
    | stored in the DataDirectory.  As memory usage gets closer to the
    | Max memory mark, the amount of message kept in the memory cache
approaches 0.
-->
<mbean code="org.jboss.mq.server.MessageCache"
      name="jboss.mq:service=MessageCache">
  <attribute name="HighMemoryMark">500</attribute>
  <attribute name="MaxMemoryMark">600</attribute>
  <depends optional-attribute-
name="CacheStore">jboss.mq:service=CacheStore</depends>
</mbean>
<!--
    | The CacheStore decides where to store JBossMQ message that
    | that the MessageCache has decided to move in secondary storage.
-->
<mbean code="org.jboss.mq.pm.file.CacheStore"
      name="jboss.mq:service=CacheStore">
  <attribute name="DataDirectory">tmp/jbossmq</attribute>
</mbean>
<!--
    | The StateManager is used to keep JMS persistent state data.
    | For example: what durable subscriptions are active.
-->
<mbean code="org.jboss.mq.sm.file.DynamicStateManager"
      name="jboss.mq:service=StateManager">
  <!-- This file is pulled from the configuration URL of the server
      -->
  <attribute name="StateFile">jbossmq-state.xml</attribute>
</mbean>

<!-- The PersistenceManager is used to store messages to disk. -->
<mbean code="org.jboss.mq.pm.file.PersistenceManager"
      name="jboss.mq:service=PersistenceManager">
  <attribute name="DataDirectory">db/jbossmq/file</attribute>
  <depends optional-attribute-
name="MessageCache">jboss.mq:service=MessageCache</depends>
</mbean>

```

```

<!--
    | The jdbc2 PersistenceManager is the new improved JDBC
implementation.
    | This implementation allows you to control how messages are
stored in
    | the database.
    | Use this PM if you want the reliability a relational database can
offer
    | you. The default configuration is known to work with hsqldb,
other databases
    | will require teaking of the SqlProperties.
-->
<!--
<mbean code="org.jboss.mq.pm.jdbc2.PersistenceManager"
      name="jboss.mq:service=PersistenceManager">
  <depends optional-attribute-
name="MessageCache">jboss.mq:service=MessageCache</depends>
  <depends optional-attribute-
name="DataSource">jboss.jca:service=LocalTxDS,name=hsqldbDS</depends>
  <depends>jboss.jca:service=LocalTxCM,name=hsqldbDS</depends>
  <attribute name="SqlProperties">
    BLOB_TYPE=OBJECT_BLOB
    INSERT_TX = INSERT INTO JMS_TRANSACTIONS (TXID) values(?)
    INSERT_MESSAGE = INSERT INTO JMS_MESSAGES (MESSAGEID,
DESTINATION, MESSAGEBLOB, TXID, TXOP) VALUES(?,?,?, ?, ?)
    SELECT_ALL_UNCOMMITTED_TXS = SELECT TXID FROM JMS_TRANSACTIONS
    SELECT_MAX_TX = SELECT MAX(TXID) FROM JMS_MESSAGES
    SELECT_MESSAGES_IN_DEST = SELECT MESSAGEID, MESSAGEBLOB FROM
JMS_MESSAGES WHERE DESTINATION=?
    SELECT_MESSAGE = SELECT MESSAGEID, MESSAGEBLOB FROM JMS_MESSAGES
WHERE MESSAGEID=? AND DESTINATION=?
    MARK_MESSAGE = UPDATE JMS_MESSAGES SET (TXID, TXOP) VALUES(?,?)
WHERE MESSAGEID=? AND DESTINATION=?
    DELETE_ALL_MESSAGE_WITH_TX = DELETE FROM JMS_MESSAGES WHERE
TXID=?
    DELETE_TX = DELETE FROM JMS_TRANSACTIONS WHERE TXID = ?
    DELETE_MARKED_MESSAGES = DELETE FROM JMS_MESSAGES WHERE TXID=?
AND TXOP=?
    DELETE_MESSAGE = DELETE FROM JMS_MESSAGES WHERE MESSAGEID=? AND
DESTINATION=?
    CREATE_MESSAGE_TABLE = CREATE TABLE JMS_MESSAGES ( MESSAGEID
INTEGER NOT NULL, \
      DESTINATION VARCHAR(50) NOT NULL, TXID INTEGER, TXOP CHAR(1),
\
      MESSAGEBLOB OBJECT, PRIMARY KEY (MESSAGEID, DESTINATION) )
    CREATE_TX_TABLE = CREATE TABLE JMS_TRANSACTIONS ( TXID INTEGER )
  </attribute>
</mbean>
-->

```

```
<!-- ===== -->
<!-- System Destinations -->
<!-- ===== -->
<!-- Dead Letter Queue -->
<mbean code="org.jboss.mq.server.jmx.Queue"
      name="jboss.mq.destination:service=Queue,name=DLQ">
  <depends optional-attribute-
name="DestinationManager">jboss.mq:service=DestinationManager</depends>
  <depends optional-attribute-
name="SecurityManager">jboss.mq:service=SecurityManager</depends>
</mbean>

</server>
```

Usamos um arquivo **xxxx-service.xml** para registrar filas de serviços do JMS e seus **“Destinations”** (Queues e Topics) na JNDI do Servidor JBOSS.

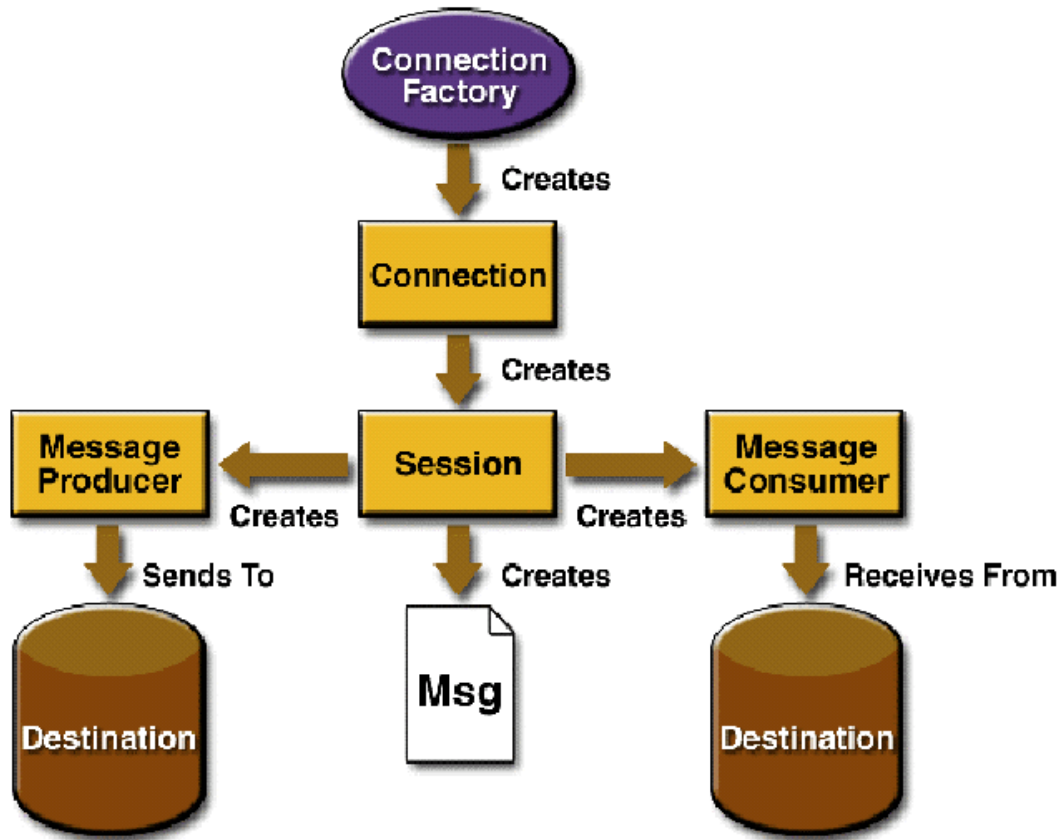
```
<?xml version="1.0" encoding="UTF-8"?>
<server>

  <mbean code="org.jboss.mq.server.jmx.Queue"
        name="jboss.mq.destination:service=Queue,name=SampleQueue">
    <depends optional-attribute-
name="DestinationManager">jboss.mq:service=DestinationManager</depends>
    <depends optional-attribute-
name="SecurityManager">jboss.mq:service=SecurityManager</depends>
    <attribute name="SecurityConf">
      <security>
        <role name="guest" read="true" write="true"/>
        <role name="publisher" read="true" write="true"
create="false"/>
        <role name="noacc" read="false" write="false" create="false"/>
      </security>
    </attribute>
  </mbean>

  <mbean code="org.jboss.mq.server.jmx.Topic"
        name="jboss.mq.destination:service=Topic,name=SampleTopic">
    <depends optional-attribute-
name="DestinationManager">jboss.mq:service=DestinationManager</depends>
    <depends optional-attribute-
name="SecurityManager">jboss.mq:service=SecurityManager</depends>
    <attribute name="SecurityConf">
      <security>
        <role name="guest" read="true" write="true"/>
        <role name="publisher" read="true" write="true"
create="false"/>
        <role name="noacc" read="false" write="false" create="false"/>
      </security>
    </attribute>
  </mbean>

</server>
```

Arquitura dos Serviços e Destinos.



A partir da JNDI, obtemos a referência via um **lookup** do ConnectionFactory e do serviço através do seu nome (padrão da J2EE: jms/QueueConnectionFactory, jms/Queue e jms/TopicConnectionFactory e jms/Topic).

De posse da ConnectionFactory, obtemos uma Session e a partir dela, podemos ler e enviar mensagens.

jms/Queue	jms/QueueConnectionFactory	QueueConnection	QueueSession	Queue
jms/Topic	jms/TopicConnectionFactory	TopicConnection	TopicSession	Topic

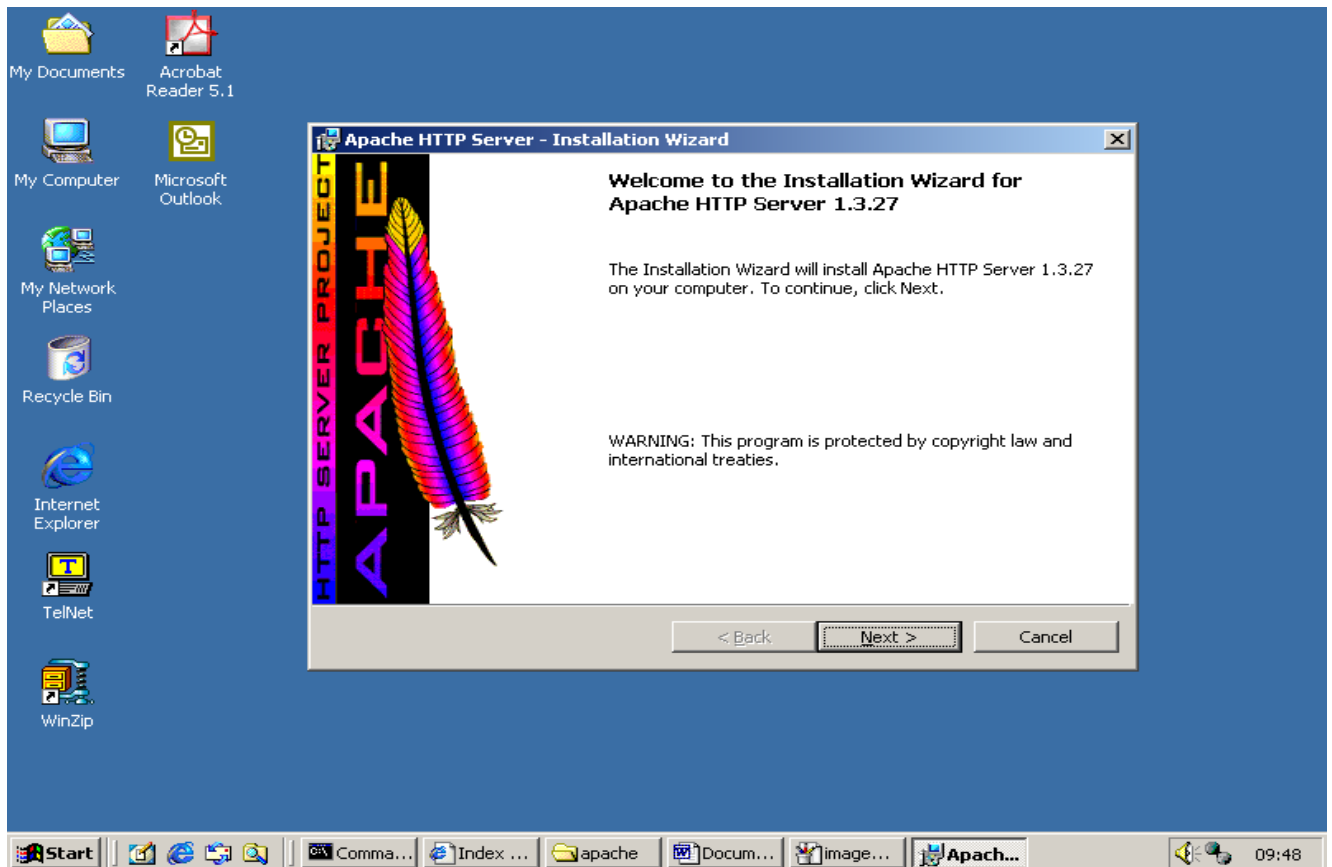
Deploy e testes de uma aplicação Web que usa o JMS

15. Instalação do Apache WebServer 1.3.X

O Apache é um WebServer mundialmente utilizado (cerca de 75% dos web hostings) e extremamente seguro e robusto.

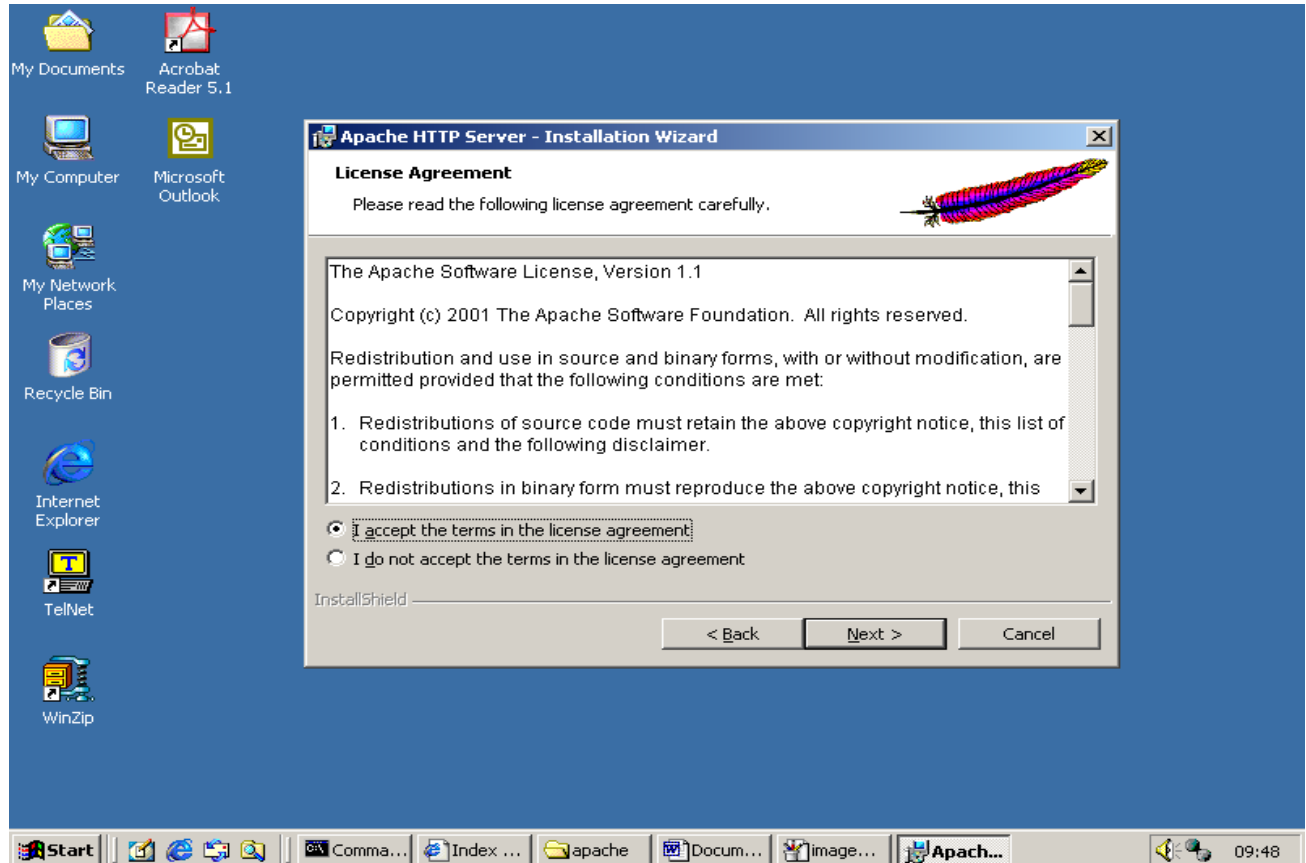
O papel de um WebServer é prover o serviço de distribuição de conteúdo estático de aplicativos web (html, imagens, mídia, outros arquivos, etc) e garantir um acesso seguro e rápido a esses conteúdos.

- Executando o Setup.
 - a. `apache_1.3.27-win32-x86-no_src.exe`

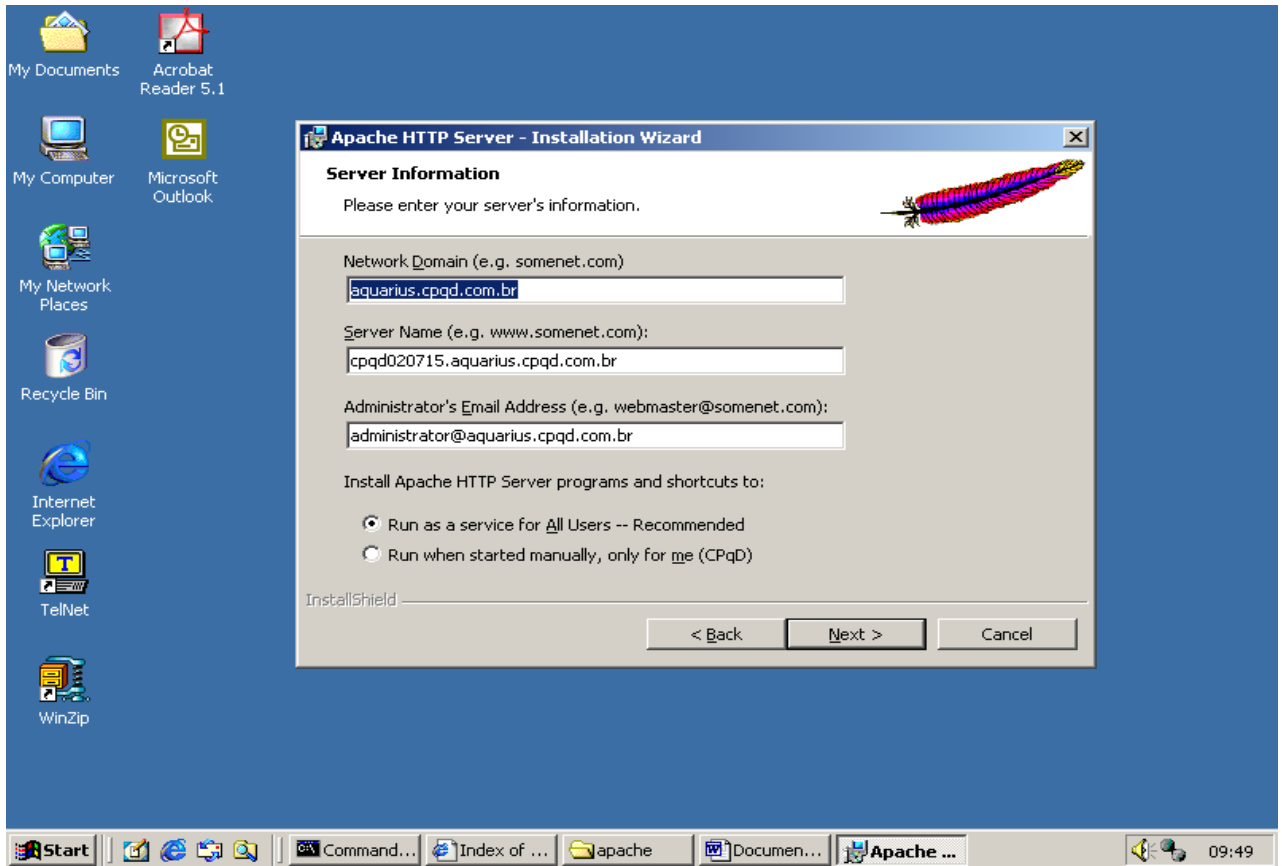


- Na tela inicial, basta continuar o processo de Setup, clicando em NEXT.
- Usar instalação completa: Server + Documentação.

- Acordar com a forma de licença de uso:

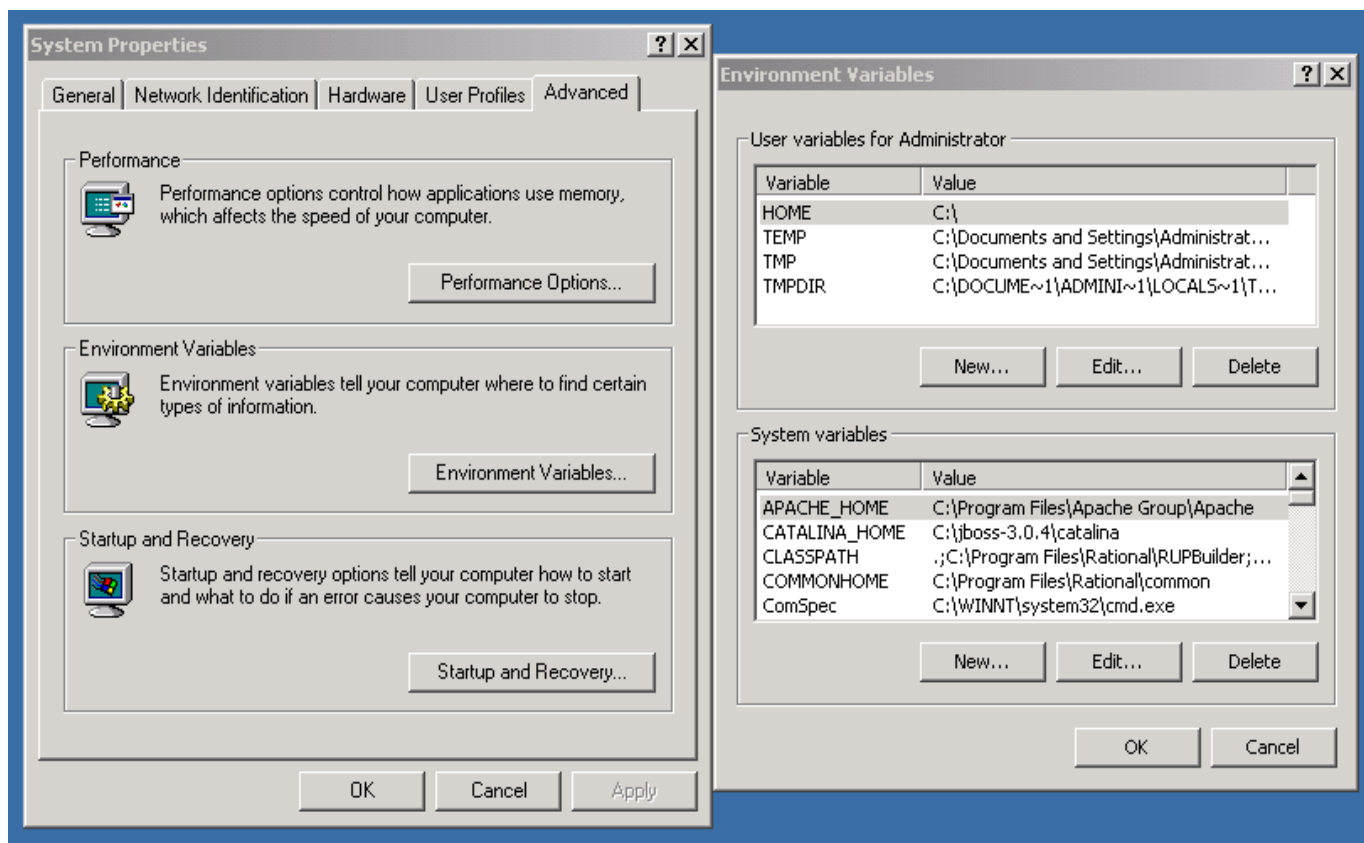


- Clicar em NEXT até chegar na tela de configuração básica;



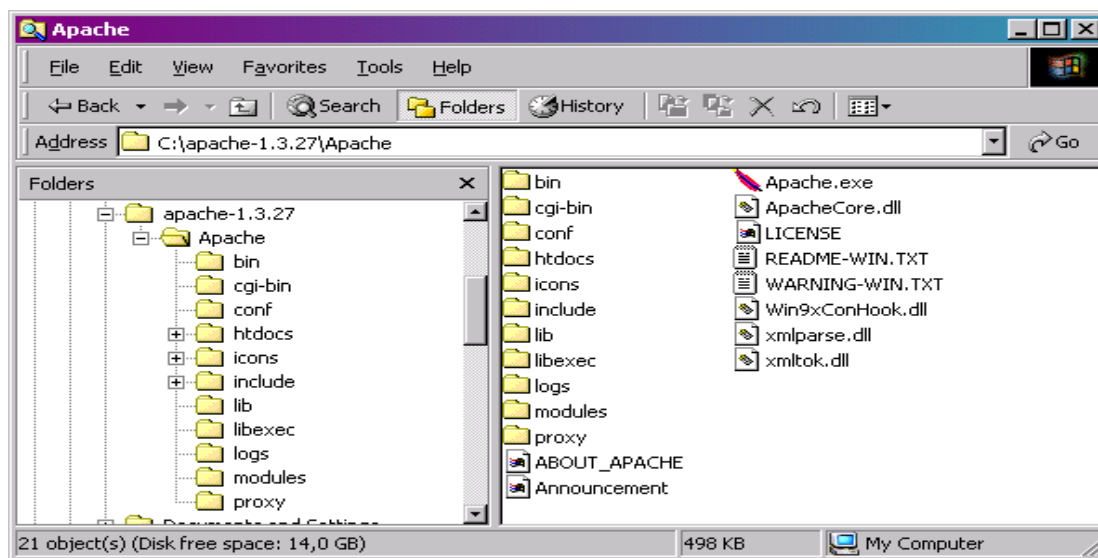
- O campo “Network Domain”, deve ser preenchido com um domínio internet, exemplos: `www.domain.com.br`, `arquarius.cpqd.com.br`, etc.
- O campo “ServerName”, recebe o nome lógico de rede pelo qual as máquinas da rede acessarão o servidor web.
- O campo “Administrator Email Adress” configura um email para notificações de eventos do servidor, exemplo: `start`, `stop`, `crash`, etc.
- A opção “Run As Service....”, facilita o gerenciamento do webserver em Produção, pois fica a cargo do Administrador configurar seu start e stop.
- A opção “Run when started.....”, facilita o uso no ambiente de Desenvolvimento, pois posso parar e iniciar seus services na hora que o usuário acha necessário.

5. Configurando o Ambiente:



Variável APACHE_HOME; VALOR <diretório de instalação do Apache>, veja o exemplo da figura acima.

Após a instalação, podemos navegar nos diretório do WebServer.

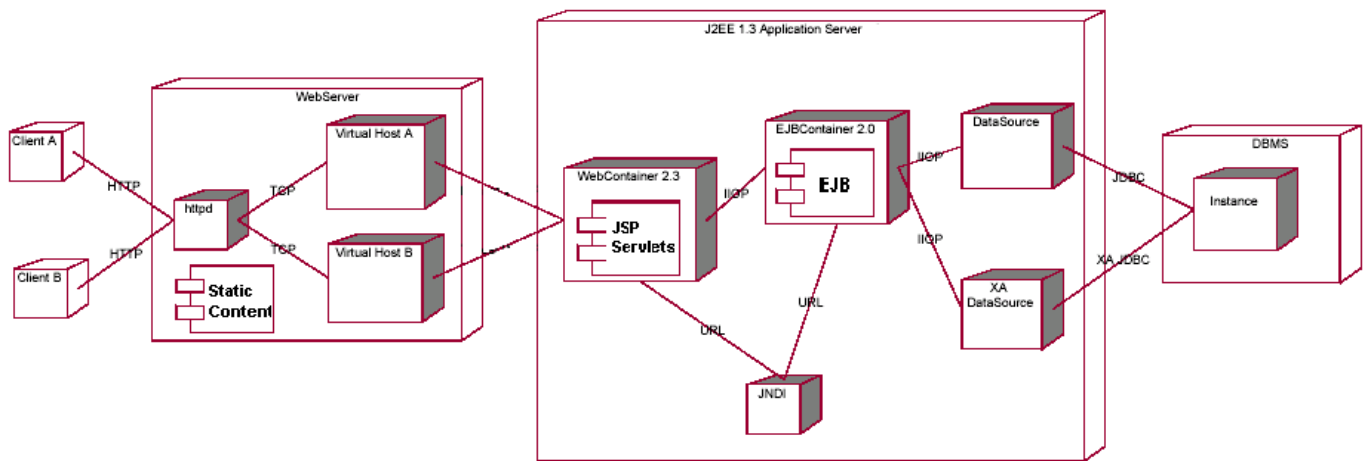


16. Integração do JBOSS+Tomcat com o Apache 1.3

A utilização de um WebServer segregado do Application Server JBOSS vai garantir a escalabilidade dos serviços do Application Server, pois vai diminuir o OverHead de IO gerado pela leitura de arquivos estáticos gerado pelas aplicações Web.

Por isso, é interessante que todos os componentes estáticos de uma aplicação Web sejam disponibilizados pelo WebServer Apache ao invés do WebServer Tomcat que está dentro do JBOSS, aumentando a robustês dos serviços.

Modelo geral da Infra-Estrutura:



Aqui vemos que podemos ter mais de um WebServer conectando ao Application Server.

E assim podemos garantir a escalabilidade da Infra-Estrutura.

Modelo da integração Apache + Tomcat.

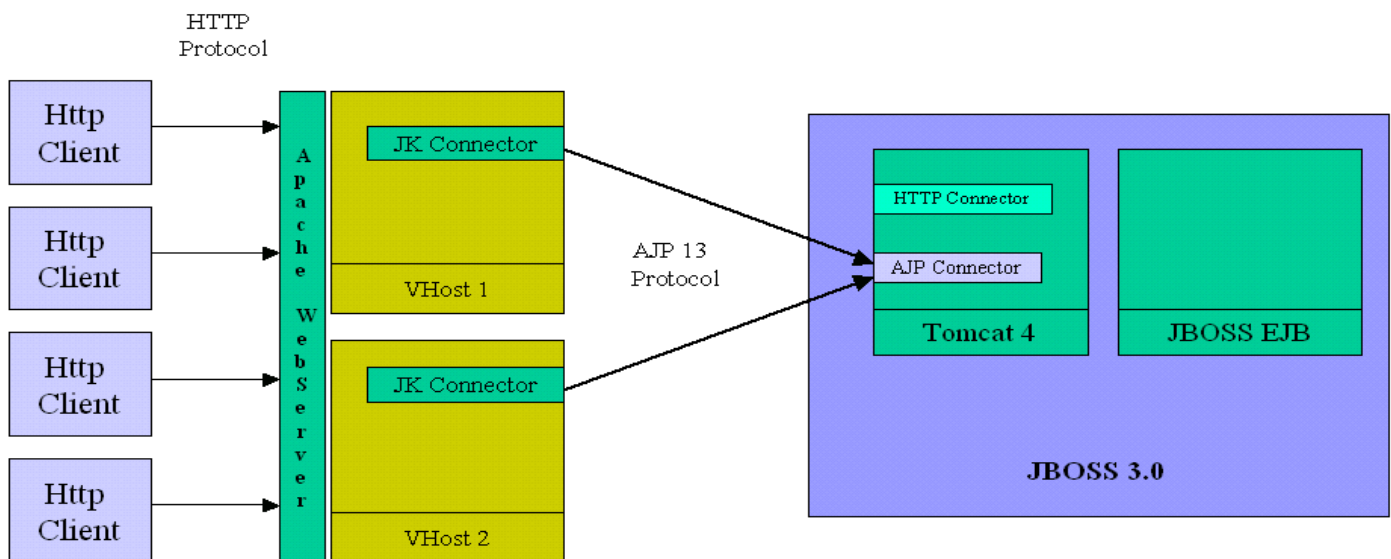
O modelo de integração é feito através de um “connector”, criado pela Apache Software com o intuito de prover uma forma fácil e robusta de integrar o Apache WebServer com o Tomcat 4.

Documentação on-line do JK Connector:

<http://jakarta.apache.org/tomcat/tomcat-4.1-doc/config/jk.html>

Toda a integração é feita através de arquivos de configuração do Apache WebServer e do Tomcat 4 (JBoss Built-In).

Arquitetura da integração usando “JK Connector”



Podemos ter mais de um Apache Virtual Host redirecionando as requisições de serviço Java para o Tomcat dentro do JBoss.

Para isso precisamos configurar:

Tomcat 4 AJP 13 Workers – criar arquivo **workers.properties**

Apache Virtual Host – configurar módulo JK e sessão de Vhosts no **httpd.conf**

JBoss 3.0 – configurar o **tomcat4-service.xml** para aceitar conexões AJP

Exemplos:

worker.properties

```
#
workers.tomcat_home=c:/jboss3.0.4
workers.java_home=c:/jbuilder7/jdk1.3.1
#WIN32
ps=\
#UNIX
#ps=/
worker.list=ajp12, ajp13

# Definition for Ajp13 worker
#
worker.ajp13.port=8009
worker.ajp13.host=cpqd050359
worker.ajp13.type=ajp13
```

alterações no httpd.conf

```
### LoadModule Section
### Tomcat Module Integration - Oziel Moreira Neto - JK
LoadModule jk_module modules/mod_jk.dll

JkWorkersFile "C:/jboss3.0.4/catalina/conf/workers.properties"
JkLogFile "C:/jboss3.0.4/server/default/log/jk.log"
JkLogLevel error
JkLogStampFormat "[%a %b %d %H:%M:%S %Y] "

## AddModule Section
### Tomcat Module Integration - Oziel Moreira Neto -JK
AddModule mod_jk.c

## VirtualHost Section
<VirtualHost cpqd050359.aquarius.cpqd.com.br>
    ServerName cpqd050359
    ServerAdmin dsbtmp1@cpdq.com.br
    ServerAlias cpqd050359 localhost
    DocumentRoot "C:/jboss3.0.4/catalina/webapps/ROOT"

    # If using JK module
    <IfModule mod_jk.c>
        JkMount /*.jsp ajp13
        JkMount /test/* ajp13
    </IfModule>

</VirtualHost>
```

tomcat4-service.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Set catalina.home to the location of the Tomcat-4.x dist.
-->
<!DOCTYPE server [
  <!ENTITY catalina.home "../catalina">
]>
<server>
  <classpath codebase="file:&catalina.home;/common/lib/" archives="*" />
  <classpath codebase="file:&catalina.home;/server/lib/" archives="*" />
  <classpath codebase="file:&catalina.home;/bin/" archives="*" />
  <classpath codebase="file:&catalina.home;/lib/" archives="*" />
  <classpath codebase="." archives="tomcat4-service.jar" />
  <mbean code="org.jboss.web.catalina.EmbeddedCatalinaServiceSX"
    name="jboss.web.service=EmbeddedCatalinaSX">
    <attribute name="CatalinaHome">&catalina.home;</attribute>
    <attribute name="Config">
      <Server>
        <Service name = "JBoss-Tomcat">
          <Engine name="MainEngine" defaultHost="cpqd050359">
            <Logger className = "org.jboss.web.catalina.Log4jLogger"
              verbosityLevel = "error" category =
                "org.jboss.web.localhost.Engine" />
            <Host name="cpqd050359" unpackWARS="true" appBase="deploy">
              <Valve className =
                "org.apache.catalina.valves.AccessLogValve"
                prefix = "cpqd050359_tomcat_access" suffix = ".log"
                pattern = "common" directory = "../server/default/log" />
              <DefaultContext cookies = "true" crossContext = "true"
                override = "true" />
            </Host>
          </Engine>
          <!-- A HTTP Connector on port 8080 -->
          <Connector className =
            "org.apache.catalina.connector.http.HttpConnector" port = "8080"
            minProcessors = "5" maxProcessors = "75" enableLookups = "false"
            acceptCount = "10" debug = "0" connectionTimeout = "60000" />

          <!-- Define an AJP 1.3 Connector on port 8009 -->
          <Connector className="org.apache ajp.tomcat4.Ajp13Connector"
            port="8009" minProcessors="5" maxProcessors="75" enableLookups =
            "false" acceptCount="10" debug="0" connectionTimeout = "60000" />

        </Service>
      </Server>
    </attribute>
  </mbean>
</server>
```

Configurando a Integração do Apache 1.3 com Tomcat 4 + JBOSS