UZORG

JBoss

Het Open Source Web OS

(NLUUG versie)

Auteur: Jos Visser

Versie : 0.1

Datum: 14-04-2003

Inhoudsopgave

1 Inleiding

"You can learn new things at any time in your life if you're willing to be a beginner. If you actually learn to like being a beginner, the whole world opens up to you."

Barbara Sher
Author of "I Could Do Anything If I Only Knew What It Was"

'Het is een goede gewoonte om te beginnen bij het begin. Maar waar is het ideale begin van de behandeling van een complex onderwerp als de JBoss applicatieserver? Voor ieder te kiezen startpunt lijkt een aanloopje te liggen dat je wellicht beter mee kan nemen om te voorkomen dat het publiek het idee heeft ergens halverwege op het parcours te zijn gedumpt met de neus in een willekeurige richting. Het verdient dan wellicht aanbeveling om nog net even wat meer te behandelen, om nog wat meer basisinformatie in de cocktail te roeren.

Het op die wijze eindeloos teruggaan in de te behandelen stof heeft echter tot gevolg dat we voor we het weten aan het begin der tijd staan en de gehele evolutie moeten behandelen omdat alleen op die wijze volledig kan worden begrepen hoe het zo gekomen is. En alhoewel er in drie kwartier evenveel tijd zit als in de hele eeuwigheid (namelijk een aftelbaar oneindig aantal tijdquanta) is het wellicht toch wat onhandig om te proberen de hele geschiedenis der dingen de revue te laten passeren alleen omdat ik als auteur graag volledig wil zijn.

Dus dient er een startpunt te worden gekozen. Dat startpunt is natuurlijk niet geheel willekeurig maar het gevaar bestaat dat ik dingen bekend acht die nog onbekend zijn, kennis verwacht waar die nog niet is of ervaring veronderstel waar alleen nog een positieve grondhouding bestaat. Ik bied daar op voorhand mijn verontschuldingen voor aan.

Deze paper gaat over JBoss, de open source J2EE applicatieserver. Aan de orde komen de interne architectuur en het gebruik van JBoss. Hierbij ga ik zowel in op het beheer van een JBoss omgeving als op het gebruik van JBoss door Java applicatieprogrammeurs. Daarbij ontkom ik er natuurlijk niet aan om op enige wijze aandacht te besteden aan Java en J2EE. Dit paper is echter geenszins een Java of J2EE tutorial. De lezer dient enige kennis van Java te hebben om te doorgronden waar relevante Java en J2EE concepten aansluiten bij (dan wel geïmplementeerd worden door) JBoss.

Commentaar, vragen en kritiek incasseer ik graag op josv@osp.nl.

1.1 Over de auteur

Jos Visser (<josv@osp.nl>) is werkzaam als senior consultant op het gebied van alles wat hij leuk vindt bij Open Solution Providers (OSP) (http://www.osp.nl) te Diemen. Gelukkig voor OSP bevat de verzameling van dingen die hij leuk vindt onderwerpen als Unix, Linux, C, Perl, Java, Clustering, TCP/IP, beveiliging en encryptie. Jos is auteur van diverse open source software, waaronder ProxyTunnel en OpenComal. Zijn persoonlijke hoekje van Cyberspace is te vinden op http://www.josvisser.nl.

Een onafhankelijke waarneemster beschreef Jos als volgt: "*Jos is very funny and is really cool!*" (Leonie Horne; 10 jaar oud).

2 Applicatieservers

Hetgeen er geweest is, hetzelve zal er zijn,
en hetgeen er gedaan is, hetzelve zal er gedaan worden;
zodat er niets nieuws is onder de zon.

Is er enig ding, waarvan men zou kunnen zeggen:
Ziet dat, het is nieuw? Het is alreeds geweest in de eeuwen,
die voor ons geweest zijn.

Prediker 1; 9-10

Alvorens onszelf in een diepgaande behandeling van JBoss te verliezen is het wellicht pedagogisch verantwoord om eerst eens de volgende vraag proberen te beantwoorden:

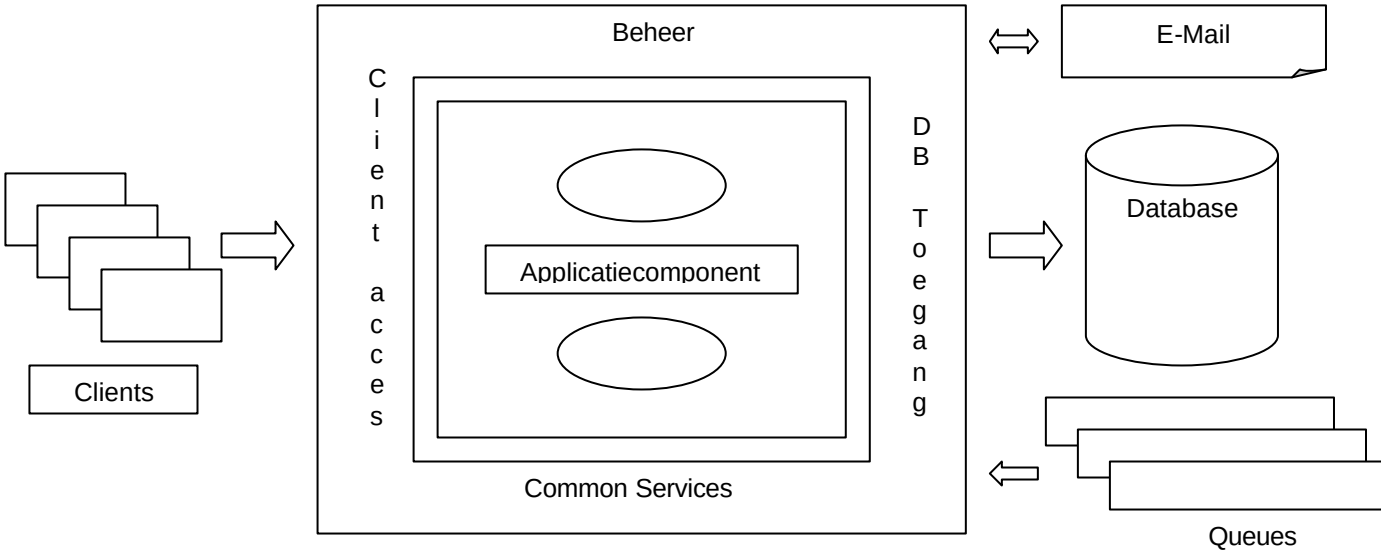
2.1 Wat is een applicatieserver?

Zo gesteld is dat best een moeilijke vraag. De woorden “applicatie” en “server” betekenen namelijk nogal wat verschillende dingen voor verschillende mensen, en de samentrekking van beide termen neemt helaas weinig van die onduidelijkheid weg. Er is helaas weinig overeenstemming over wat een applicatieserver precies is en welke diensten een appserver feitelijk levert. Een rondgang langs de diverse leveranciers van applicatieservers laat de nietsvermoedende onderzoeker met het gevoel achter dat een applicatieserver een stuk software is die de problemen oplost die je hebt als je geen applicatieserver toepast…

Ik zou zelf een applicatieserver willen omschrijven als een stuk infrastructurele software die tot doel heeft applicatiecomponenten te herbergen en die allerlei services aan die applicatiecomponenten aanbiedt. De gedachte achter het concept van de applicatieserver is dat allerlei zaken zoals database access, interactie met user interfaces, beveiliging en schaalbaarheid infrastructureel (buiten de applicatie) op te lossen zijn zodat de programmeur zich alleen nog maar bezig hoeft te houden met het programmeren van de “business logica” (op zich al een sterk aan inflatie onderhevige kreet).

2.2 Applicatieserver functionaliteit

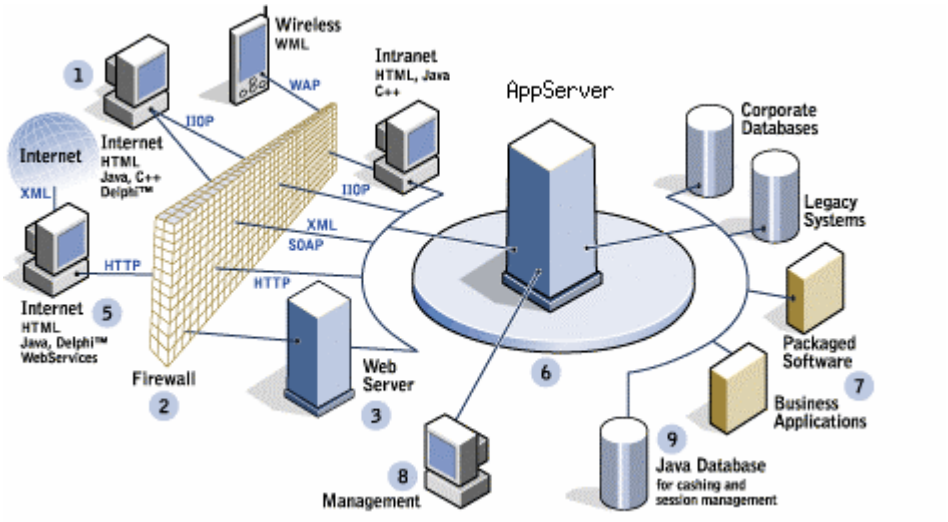
Een applicatieserver kan dus het best worden gezien als een container voor applicatiecomponenten:



De applicatiecomponenten die onder controle van de applicatieserver draaien interacteren via hun omgeving met allerlei andere elementen in de infrastructuur. Hierbij draagt de applicatieserver zorg voor de daadwerkelijke uitvoering van die interactie. Dit dient een aantal doelen:

- De applicatieprogrammeur hoeft zich geen/minder zorgen te maken over de feitelijkheden van de communicatie met de omgeving.
- Er is een functioneel rijke omgeving van standaarddiensten aanwezig waar de programmeur zich geen zorgen meer over hoeft te maken.
- De feitelijk communicatieprotocollen worden door de applicatieserver geïmplementeerd. Hierdoor kunnen we na ontwikkeling van de software nog veranderingen maken in bijvoorbeeld welk merk database wordt gebruikt of wat het communicatieprotocol met de clients is (IIOP, JRMP, HTTP).
- Omdat de applicatieserver de controle over de omgeving heeft kunnen zaken als clustering en load balancing transparant door de applicatieserver worden geregeld.

Er wordt getracht een omgeving te implementeren waarbij de applicatieprogrammeur zich alleen nog maar zorgen hoeft te maken over het programmeren van de business logica. De applicatieserver regelt de rest. Dit wordt in het onderstaande plaatje nog eens weergegeven:



2.3 Een nieuw verschijnsel?

Applicatieservers zijn geenszins een nieuw verschijnsel te noemen. Verre van dat zelfs! Op IBM mainframes (die op zich al sinds het begin der tijden aanwezig zijn) wordt al sinds jaar en dag gebruik gemaakt van CICS en IMS/DC. Ook in de UNIX omgeving is de applicatieserver geen onbekende. Pakketten als CICS/6000 (AIX) en Tuxedo bieden daar al jaren diensten die tegenwoordig onder de noemer “applicatieserver” zouden moeten worden geschaard.

3 Java 2 Enterprise Edition (J2EE)

“Ingenuity, plus courage, plus work, equals miracles”

Bob Richards
American Olympic Pole Vaulting Champion

Met het verschijnen van Java aan het programmeertaal (pardon “omgeving”) front is er weer eens frisse wind langs datzelfde front gewaaid. Java voegde niets toe aan wat er al was, maar verpakte allerlei populaire verschijnselen weer eens in een nieuw jasje. Aldus toegerust met het beste/handigste wat de mensheid zo de laatste dertig jaar op het gebied van programmeren had verzonnen bleek Java een groot en gapend gat in te vullen. Waar Java eerst nog per ongeluk werd gezien als een programmeertaal voor interactieve web applicaties (de vermalijde “applets”) werden we het er gelukkig al snel over eens dat Java toch meer bedoeld was voor serieuze client/server applicaties.

Er valt heel wat over (zowel voor als tegen) Java te zeggen, maar ik zou graag willen volstaan met de opmerking dat het een goede, overzichtelijke en complete programmeertaal is die allerlei interessante technologieën binnen het handbereik van de “gewone” programmeur heeft gebracht.

3.1 De Java omgeving

Een volledige standaard Java omgeving bestaat uit de volgende onderdelen:

- De Java Virtual Machine (JVM) ; interpreteert de Java byte code (Java machinetaal).
- De Java compiler; vertaalt Java sources naar byte code.
- De Java API; een uitgebreide standaardbibliotheek met algemeen (herbruikbare) klassen.

Deze standaard Java omgeving (de Java2 Standard Edition: J2SE) komt tot ons in twee varianten:

- De Java Runtime Environment (JRE ; JVM en API implementatie).
- De Java Development Kit (JDK ; JRE plus compiler en API sources).

De referentieimplementatie van Java wordt geleverd door Sun Microsystems (voor de Solaris, Windows en Linux platformen). Daarnaast zijn er ports van die referentieimplementatie naar andere platformen (b.v. HP-UX) en zijn er alternatieve implementaties die zich volledig aan de Java standaard (trachten te) houden.

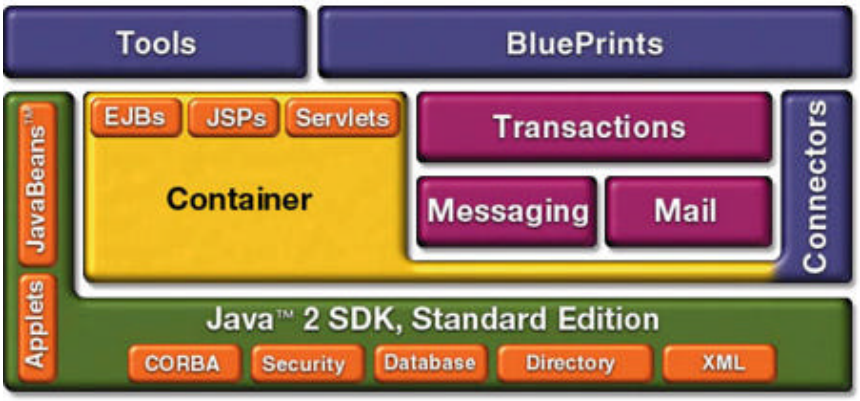
De J2SE is uitermate geschikt voor het ontwikkelen van (grafische) client applicaties en bevat onder andere ook onderdelen voor het benaderen van databases (JDBC; Java Database Connectivity).

3.2 Enterprise applicaties

Voor het ontwikkelen van “Enterprise applicaties” is de standaard Java API niet afdoende. Om die reden heeft Sun Microsystems (in samenwerking met geïnteresseerde partijen) de Java2 Enterprise Edition (J2EE) standaard ontwikkeld. J2EE is een verzameling aan definities en (referentie)implementaties van Java onderdelen die nodig zijn om complexe gedistribueerde applicaties te bouwen. Voorbeelden van onderdelen van J2EE zijn:

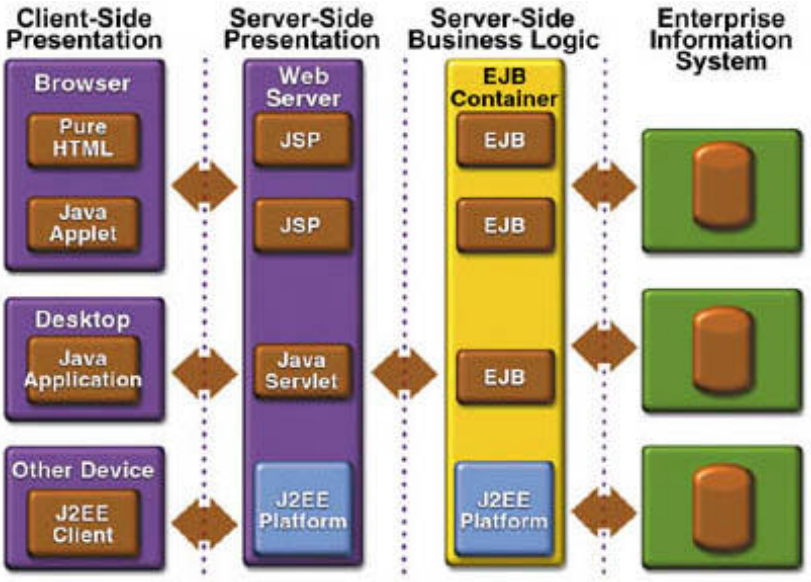
- Java Server Pages (JSP); een standaard voor HTML pagina’s waarin inline Java code is opgenomen (voor verwerking door een web server).
- JavaMail; een interface tussen Java componenten en e-mail systemen.
- Java Naming and Directory Interface (JNDI); een specificatie over hoe vanuit Java programma’s directories van informatie te benaderen (b.v. LDAP, Active Directory, DNS).
- Enterprise Java Beans (EJB); de Java2 technologie voor gedistribueerde componenten die in een applicatieserver draaien.
- Java Messaging Standard (JMS); het interface tussen Java programma’s en “message queues” (zoals IBM MQueue Series en TibCo).

Het volgende plaatje geeft een grafisch overzicht van de structuur van J2EE :



3.3 J2EE applicatie architectuur

J2EE tekent een vrij scherp beeld van hoe (gedistribueerde) Java Enterprise Applicaties in elkaar dienen te zitten:



Het is echter bij keuze voor J2EE niet strikt noodzakelijk om voor precies dit model te kiezen. De J2EE technologieën kunnen naar keuze worden gebruikt en gemengd.

3.4 J2EE interfaces

J2EE is voor een groot gedeelte “alleen maar” een specificatie en implementatie van het interface tussen Java programma’s en de “service providers” (zoals databases, appservers, queueing systemen, mail systemen, directories). De feitelijke implementaties van de services dienen te worden geleverd door derde partijen.

Naast het feitelijke interface tussen een Java programma en een J2EE service specificeert de J2EE standaard ook hoe een Java programma een service provider kan lokaliseren en een verbinding met (een instance van) die service provider kan maken. Het is tamelijk eenvoudig om de parameters waarmee de service provider kan worden gelokaliseerd in een externe configuratie (b.v. een property file) op te nemen zodat er tijdens uitvoering nog kan worden gewisseld van feitelijke service implementatie.

De wijze waarop dit (dus) is geïmplementeerd zorgt ervoor dat een J2EE applicatie volledig onafhankelijk kan worden gemaakt van de feitelijke J2EE (service provider) omgeving. De belofte die dit in zich biedt is dat J2EE applicaties zonder aanpassingen in compleet verschillende J2EE omgevingen (met andere databases, containers, queueing/mail systemen) kunnen worden gedraaid zonder de applicatie aan te passen! Een belofte is een mooi ding…

3.5 De rol van de applicatieserver in J2EE

In de J2EE architectuur is aan de applicatieserver de rol van het “J2EE platform” toebedeeld (zie ook bovenstaande plaatje). De appserver is de EJB container (waarbinnen de applicatiecomponenten draaien) en de web container (waarbinnen servlets en Java Server Pages) draaien. Daarnaast implementeert de appserver de service providers voor J2EE componenten of biedt (conform de J2EE specificatie) toegang tot service providers van derde partijen.

4 JBoss

"If you would have a faithful servant, and one that you like, serve yourself."

Benjamin Franklin
1706-1790, American Scientist, Publisher, Diplomat

JBoss is een open source J2EE compliant applicatieserver. Laten we allereerst deze beschrijving even uit elkaar trekken om zodoende snel op de hoogte te geraken van wat JBoss is:

JBoss is geheel Open Source en wordt gedistribueerd onder de Lesser GNU Public License (LGPL). De LGPL maakt het mogelijk om JBoss onderdelen te gebruiken als componenten in een closed source applicatie. Gezien de specifieke methode waarop J2EE applicatiecomponenten in de applicatieserver worden gehangen (dynamisch laden gedurende uitvoering en vervolgens rechtstreeks aanroepen van de componentmethodes) is de LGPL de enige methode om openheid van de source code te garanderen zonder de bruikbaarheid van de appserver voor closed source applicaties te garanderen.

(Ter vergelijking: het toepassen van de GPL-licentie zou waarschijnlijk tot gevolg hebben gehad dat alleen open source applicaties JBoss hadden kunnen gebruiken en het toepassen van de BSD-licentie zou hebben betekend dat verbeteringen aan JBoss closed source zouden kunnen worden gemaakt.

De organisatie rondom JBoss is de JBoss Group LLC, een "Limited Liability Corporation" (een soort besloten vennootschap) onder leiding van de JBoss projectleider Marc Fleury. De kernontwikkelaars van JBoss staan op de loonlijst van de JBoss Group en verdelen hun tijd tussen het ontwikkelen van JBoss, het schrijven van documentatie en het uitvoeren van consultancy en trainingswerkzaamheden. De JBoss Group wordt aldus gefinancierd door de inkomsten uit het verkopen van de documentatie en de consultancy/trainingen.

Persoonlijk denk ik dat dit verstandshuwelijk tussen open source en commercie een solide basis vormt voor het vrij beschikbaar zijn en blijven van JBoss.

De ontwikkelaars van JBoss ontwikkelen met als doel om volledig compliant te zijn met de Java2 Enterprise Edition (J2EE) specificaties. Op moment van schrijven heeft de JBoss Group bij Sun Microsystems een aanvraag ingediend om JBoss officieel te laten certificeren als zijnde in overeenstemming met J2EE.

4.1 JBoss features overzicht

JBoss is een volledige en functionele applicatieserver voor J2EE componenten. Zoals in een eerder hoofdstuk in dit paper uit de doeken is gedaan is J2EE een "lege" specificatie (van interfaces) die

grotendeels door de appserver leverancier moet worden aangevuld met specifieke implementaties van de in de specificatie benoemde interfaces. Daarnaast is het de appserver leverancier natuurlijk toegestaan om faciliteiten toe te voegen aan de appserver. Maak je daar echter als applicatieprogrammeur gebruik van dan “breek” je natuurlijk de J2EE compatibiliteit en kan dit je mogelijkerwijs verhinderen om je applicatie naar andere J2EE appservers te porten.

In vogelvlucht bezien bevat JBoss de volgende features:

1. EJB (Enterprise Java Beans) 2.0 server/container met ondersteuning voor Container Managed Persistence (CMP), Entity Beans, Session Beans, Message Driven Beans en transacties.
2. Een Java Naming en Directory Interface (JNDI) component voor het ondervragen van geregistreerde componenten en dergelijke.
3. Ondersteuning voor Servlets en Java Server Pages door integratie met Jakarta (Apache) Tomcat of Jetty.
4. Een JMS compatible queue manager (JBossMQ) met ondersteuning voor queues en topics (publish/subscribe).
5. Integratie met JAAS (Java Authentication and Authorisation Service) voor beveiliging.
6. Interne architectuur opgebouwd volgens de Java Management Extensions (JMX) framework.
7. Ingebouwde ondersteuning voor High Availability en schaalbaarheid via clustering van JBoss instances.
8. Ondersteuning voor CORBA IIOP (JBoss 3.2.0)

4.2 Download

JBoss kan worden gedownload van de JBoss pagina’s bij SourceForge:

http://sourceforge.net/projects/jboss

Op moment van schrijven is JBoss 3.0.6 het meest recente stabiele release en is JBoss 3.2.0 release candidate 4 (3.2.0RC4) ook beschikbaar voor hen die op de “bleeding edge” wensen te leven.

Het stabiele release is beschikbaar in drie pakketten:

jboss-3.0.6-src.tgz	19 MB	JBoss source
jboss-3.0.6.zip	27 MB	JBoss binary package met Jetty geïntegreerd
jboss-3.0.6_tomcat-4.1.18.zip	32 MB	JBoss binary package met Tomcat 4.1.18 geïntegreerd.

De sources vereisen in principe alleen Apache Ant om te worden gebouwd (Ant is een Java make/build tool; zie http://ant.apache.org).

4.2.1 Jetty versus Tomcat

Het verschil tussen de Jetty en de Tomcat versie van JBoss is welk pakket er wordt gebruikt als web server c.q. Java servlet/JSP provider.

Jetty (http://jetty.mortbay.org/jetty/index.html) is een open source web server en servlet/JSP container ontwikkeld door (c.q. onder auspiciën van) Mort Bay Consulting.

Tomcat (http://jakarta.apache.org/tomcat/index.html) is de referentie-implementatie servlet/JSP container van de Apache Foundation. De Tomcat-versie van JBoss levert een MBean (zie later voor meer informatie over MBeans) mee die Tomcat start en laadt in dezelfde JVM als waar JBoss draait.

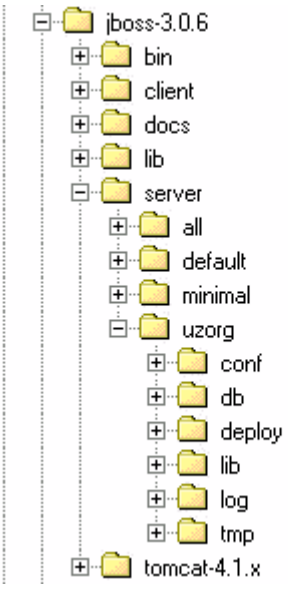
Mijn voorkeur gaat er in principe naar uit om de Tomcat variant van JBoss te gebruiken. De reden daartoe is eenvoudigweg dat ik vaker Tomcat heb gebruikt in diverse producten en daar altijd blij en gelukkig mee ben geweest.

Indien gewenst kan overigens voor zowel Tomcat als Jetty in een JBoss omgeving nog Apache worden gehangen. In die situaties worden de statische content requests (HTML, JS, GIF, JPG en dergelijke) door Apache opgelost en worden alleen de servlet/JSP verzoeken naar JBoss (Jetty of Tomcat) doorgegeven.

4.3 Installatie

Iedereen die het “unzip” commando machtig is kan JBoss installeren. De JBoss installatie vereist verder geen aanpassingen van de systeemconfiguratie of andere modificaties.

Het JBoss archief explodeert in een directory tree met ongeveer de volgende layout:



Deze directories bevatten de volgende componenten:

bin			JBoss start scripts (run.sh)
client			De Java jars (libraries) die op een JBoss client beschikbaar moeten zijn (o.a. de JNDI client, de EJB RMI clients, de JBoss MQ client et cetera.
docs			Wat documentatie en voorbeeld bestanden. De echt intelligente documentatie dient echter separaat te worden aangeschaft.
lib			Algemeen benodigde Java jars (libraries) zoals GNU regexp, Xerces (XML parser) en JBoss algemene libraries
server			De subdirectory waaronder de server configuraties leven. Bij het starten van JBoss dient een server configuratienaam te worden opgegeven en die naam dient als subdirectory voor te komen in de $JBOSS_HOME/server directory.
	all		De standaard server configuratie die alle JBoss componenten activeert.
	default		De server configuratie die wordt geactiveerd indien geen expliciete JBoss configuratie wordt aangegeven bij het starten.
	minimal		De standaard server configuratie die geen enkele JBoss component activeert.
	uzorg		Een eigenhandig samengestelde (custom configuratie) voor gebruik van JBoss.
		conf	Bevat configuratiebestanden voor JBoss (en applicatie) componenten.
		db	De standaard directory voor (HyperSonic SQL) tabellen en JBossMQ persistentie bestanden.
		deploy	De directory met descriptors van de componenten die moeten worden geactiveerd (zowel JBoss als applicatiecomponenten).
		lib	De Java jars met de code voor (JBoss) componenten die in deze server configuratie moeten worden geactiveerd.
		log	De JBoss logbestanden (Log4J).

		tmp	Tsja….. ✍
tomcat-4.1.x			De base directory van de (geïntegreerde) Tomcat installatie. De directory structuur van deze subdirectory komt overeen met die van een standaard Tomcat installatie.

4.4 Uitvoeren

Om JBoss uit te kunnen voeren is een Java 1.4 (compatible) SDK (JDK) nodig. In principe is JBoss 100% Pure Java en kan dus op ieder Java compatible platform worden uitgevoerd. Onder Linux heb ik ervaring met zowel de Sun als de Blackdown JDK 1.4.

De aanbevolen wijze om JBoss te starten is naar de JBoss bin directory te verplaatsen en daar het "run.sh" script te starten. Bij het starten vanuit een andere locatie wil nog wel eens probleempje optreden met betrekking tot relatieve padnamen waarnaar bijvoorbeeld vanuit de configuratie of deploy bestanden wordt verwezen:

```
$ cd ~/jboss/jboss-3.0.6/bin
$ export JAVA_HOME=/opt/java1.4 # por ejemplo
$ ./run.sh –c uzorg
```

Via de "-c" optie wordt aangegeven welke server configuratie dient te worden geactiveerd. Na het opkomen van de JBoss kernel wordt overgeschakeld naar de "deploy" directory van de genoemde configuratie en alle onderdelen die in die directory aanwezig zijn(zie verderop in dit paper) worden geactiveerd. Indien geen "-c" wordt meegegeven wordt de "default" configuratie geactiveerd.

Standaard staat JBoss geconfigureerd om alle logging naar zowel het console als naar de file "log/server.log" te schrijven. Die logging kan naar volledig eigen inzicht worden gewijzigd door het aanpassen van "conf/log4j.xml".

Na verloop van tijd verschijnt de navolgende kreet in de log en is JBoss "ready for e-business" (om maar eens een WebSphere kreet te misbruiken):

```
2003-04-07 16:22:43,102 INFO  [org.jboss.system.server.Server] JBoss (MX
MicroKernel) [3.0.6 (CVSTag=JBoss_3_0_6 Date=200301260037)] Started in 1m:56s:725ms
```

4.5 Stoppen

Om JBoss te stoppen kan gebruik worden gemaakt van het "shutdown.sh" script in de JBoss bin directory. Er zijn echter gevallen bekend ("to put it mildly") dat dit script niet functioneerde. In dat geval kan een Ctrl-C of een "kill" wonderen doen ✍. Deze beide methodes starten een ordentelijke "undeploy" van alle componenten en beëindigen de JBoss omgeving.

5 JBoss architectuur

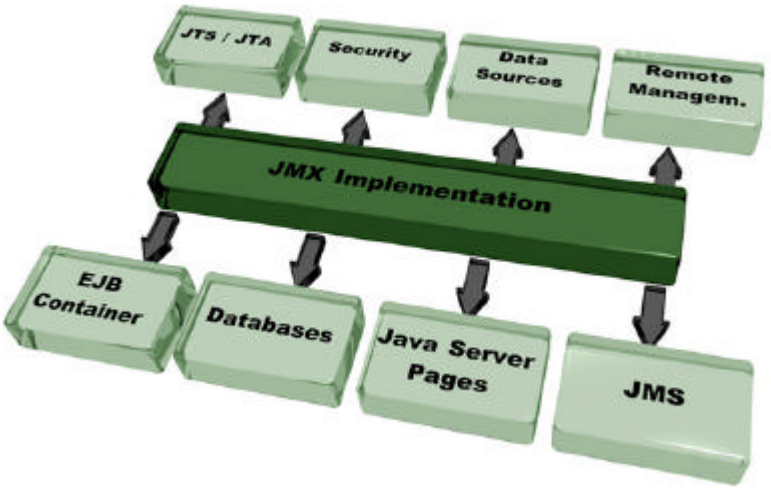
"Architecture is the art of how to waste space."

Philip Johnson
American Architect and Theorist

5.1 Java Management Extensions

JBoss is intern helemaal opgebouwd volgens de Java Management Extensions (JMX) architectuur. JMX is een (Java) standaard voor het managen en monitoren van hard- en software componenten. Alle JBoss componenten zijn geschreven als JMX componenten (ook wel "MBeans" (Management Beans) geheten". De kernel van JBoss is een JMX implementatie die zich "alleen" bezighoudt met het laden, activeren, koppelen en ontladen van dit soort MBeans.

De MBeans worden door de JBoss kernel in de "JMX bus" geschoven en kunnen op die manier met elkaar communiceren:



De JMX specificatie beschrijft de manier waarop de JMX implementatie (JBoss) en de MBeans met elkaar kunnen communiceren. JMX bevat ondersteuning voor attributen (get/set), methodes en event handling.

Er zitten grote voordelen aan het gebruik van JMX door JBoss. Enerzijds maakt het JBoss zelf compleet "pluggable" (alleen die componenten die benodigd zijn hoeven te worden geactiveerd) en anderzijds is het zonder meer mogelijk om andere (zelf ontwikkelde) MBeans in de JBoss omgeving te schuiven. Zo is bijvoorbeeld de Tomcat integratie in JBoss een "wrapper" Mbean die na activering de Tomcat start procedure uitvoert (runt dezelfde klasse die het Tomcat startup.sh script ook uitvoert).

5.2 De deploy directory

De deploy directory van de JBoss server configuratie bevat bestanden die aangeven welke JMX MBeans door de JBoss kernel moeten worden gedeployed. Als die directory leeg is (zoals bijvoorbeeld in de JBoss "minimal" server configuratie) komt de JBoss kernel op en worden er geen componenten geactiveerd. In principe moet er voor iedere feature en applicatiecomponent die we willen activeren een specificatie in de deploy directory staan.

JBoss ondersteunt "hot deploy". De JBoss deployer doorzoekt regelmatig de deploy directory en indien daar een nieuwe of gewijzigde JMX componentbeschrijving wordt gevonden wordt die ogenblikkelijk geactiveerd. Ook als een JMX component wordt verwijderd wordt de JMX omgeving ogenblikkelijk aangepast aan die actie

In onderstaande voorbeeld heb ik een volledig lege JBoss server configuratie aangemaakt (“sample”). Na het starten kopieer ik met “cp” een MBean beschrijving in de sample/deploy directory (de “counter-service.xml”). Na korte tijd begint JBoss met de deploy:

```
11:43:44,666 INFO  [Server] JBoss (MX MicroKernel) [3.0.6 (CVSTag=JBoss_3_0_6
Date=200301260037)] Started in 0m:4s:796ms
11:44:14,710 INFO  [MainDeployer] Starting deployment of package:
file:/home/josv/jboss/jboss-3.0.6_tomcat-4.1.18/server/sample/deploy/counter-
service.xml
```

Na verloop van tijd verwijder ik die file weer, wat het volgende resultaat geeft:

```
11:48:50,420 INFO  [MainDeployer] Undeploying file:/home/josv/jboss/jboss-
3.0.6_tomcat-4.1.18/server/sample/deploy/counter-service.xml
11:48:50,425 INFO  [DeploymentInfo] Cleaned Deployment:
file:/home/josv/jboss/jboss-3.0.6_tomcat-
4.1.18/server/sample/tmp/deploy/server/sample/deploy/counter-service.xml/2.counter-
service.xml
11:48:50,425 INFO  [MainDeployer] Undeployed file:/home/josv/jboss/jboss-
3.0.6_tomcat-4.1.18/server/sample/deploy/counter-service.xml
```

(de JBoss log regels kunnen nogal lang worden ✍)

Wat nog wel eens een probleem kan zijn is dat de JBoss deployer de directory doorzoekt op het moment dat de kopieeractie nog loopt. Het kan dan (met name bij grote EJB jars) voorkomen dat JBoss tracht een halve file te deployen, wat de nodige foutmeldingen oplevert. De oplossing hiervoor is natuurlijk om in plaats van een kopie een verplaatsing te doen (waarbij de hele file als atomaire actie zichtbaar wordt gemaakt in de directory).

MBeans kunnen onderling van elkaar afhankelijk zijn. De (XML) MBean specificatie kan “dependencies” beschrijven die door de JBoss deployer in acht worden genomen. In bovengenoemde voorbeeld (“counter-service.xml” deployen in een lege JBoss) gaat de deploy ook niet goed (niet getoond) in verband met het ontbreken van MBeans waar de voorbeeld service gebruik van maakt.

5.3 Deploy formaten

JBoss kent wat soorten verschillende deployers die in staat zijn om verschillende types bestanden uit de deploy directory te deployen. JBoss selecteert automatisch welke deployer te activeren op basis van de naam (en eventueel andere attributen) van de te deployen file.

5.3.1 SARDeployer

De basisdeployer van JBoss is de “SARDeployer” (SAR staat voor “Service ARchive”). Dit is een deployer die “*-service.xml” bestanden in de deploy directory verwerkt. Die SAR service XML bestanden bevatten een beschrijving van een of meer MBeans die in de JMX bus moeten worden gehangen. De MBean beschrijving in een SAR XML bestand bevat zaken als:

- De naam van de MBean klasse
- Parameters voor de MBean
- Afhankelijkheden naar andere MBeans (dependencies)
- De naam van de geïstantieerde MBean
- Het Java “class path” dat moet worden gebruikt voor deze MBean

Onderstaande voorbeeld toont de SAR XML beschrijving van de JBoss scheduler service:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE server>
<!-- $Id: scheduler-service.xml,v 1.2 2002/02/24 10:24:35 user57 Exp $ -->

<server>
```

```
  <classpath codebase="lib"
             archives="scheduler-plugin.jar,
                       scheduler-plugin-example.jar"/>

  <!-- ===================================================================== -->
  <!-- Scheduler Service                                                     -->
  <!-- ===================================================================== -->

  <!--
     | This example shows how to use a pluggable Schedulable impl

  <mbean code="org.jboss.varia.scheduler.Scheduler"
        name=":service=Scheduler">
    <attribute name="StartAtStartup">true</attribute>
    <attribute
name="SchedulableClass">org.jboss.varia.scheduler.example.SchedulableExample</attri
bute>
    <attribute name="SchedulableArguments">Schedulabe Test,12345</attribute>
    <attribute name="SchedulableArgumentTypes">java.lang.String,int</attribute>
    <attribute name="InitialStartDate">0</attribute>
    <attribute name="SchedulePeriod">10000</attribute>
    <attribute name="InitialRepetitions">-1</attribute>
  </mbean>

  -->

  <!--
     | This example shows how to use a target MBean

  <mbean code="org.jboss.varia.scheduler.example.SchedulableMBeanExample"
        name=":name=SchedulableMBeanExample">
  </mbean>
  <mbean code="org.jboss.varia.scheduler.Scheduler"
        name=":service=Scheduler,name=SchedulableMBeanExample">
    <attribute name="StartAtStartup">true</attribute>
    <attribute name="SchedulableMBean">:name=SchedulableMBeanExample</attribute>
    <attribute name="SchedulableMBeanMethod">hit( NOTIFICATION, DATE, REPETITIONS,
SCHEDULER_NAME, java.lang.String )</attribute>
    <attribute name="InitialStartDate">NOW</attribute>
    <attribute name="SchedulePeriod">10000</attribute>
    <attribute name="InitialRepetitions">10</attribute>
  </mbean>

  -->

</server>
```

5.3.2 Java archive deployers

Andere JBoss MBean deployers verwerken Java archieven (JAR files) die in de deploy directory worden aangetroffen. Deze zogenaamde “SubDeployers” verwerken JARs van de volgende types:

- Webapplicatie archieven (.war). worden losgelaten in Jetty of Tomcat.
- EJB archieven (.jar met META-INF/ejb-jar.xml descriptor), worden uit elkaar getrokken en iedere bean in het archief wordt gedeployed.
- EAR archieven (.ear)
- RAR archieven (.rar met META-INF/ra.xml descriptor; archieven met implementaties van connectors conform de Java Connector Architecture).

5.3.3 Andere deployers

De deployer architectuur van JBoss is compleet uitbreidbaar. Dit maakt het mogelijk om deployers toe te voegen die andere typen bestanden kan deployen.

Zo is bijvoorbeeld in JBoss 3.2 een deployer opgenomen die datasource MBeans kan creëren op basis van specificaties in een “*-ds.xml” bestand:

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

<!-- ===================================================================== -->
<!--                                                                       -->
<!--  JBoss Server Configuration                                           -->
<!--                                                                       -->
<!-- ===================================================================== -->

<!-- $Id: hsqldb-ds.xml,v 1.1.2.1 2002/12/12 03:07:05 jboynes Exp $ -->

<datasources>
  <local-tx-datasource>
    <depends>jboss:service=Hypersonic</depends>
    <jndi-name>DefaultDS</jndi-name>
    <connection-url>jdbc:hsqldb:hsql://localhost:1701</connection-url>
    <driver-class>org.hsqldb.jdbcDriver</driver-class>
    <connection-property name="user">sa</connection-property>
    <connection-property name="password"></connection-property>
    <min-pool-size>5</min-pool-size>
  </local-tx-datasource>

  <mbean code="org.jboss.jdbc.HypersonicDatabase"
        name="jboss:service=Hypersonic">
    <attribute name="Port">1701</attribute>
    <attribute name="Silent">true</attribute>
    <attribute name="Database">default</attribute>
    <attribute name="Trace">false</attribute>
    <attribute name="No_system_exit">true</attribute>
  </mbean>
</datasources>
```

Deze wijze van datasource specificatie is vele malen eenvoudiger dat de SAR deployer gebaseerde datasource specificatie in JBoss 3.0.6 en eerder.

5.4 JBoss configuratie

JBoss (JMX/MBean) componenten die aanvullende parameters nodig hebben worden doorgaans geconfigureerd aan de hand van bestanden in de server “conf” directory.

```
josv@jadzia:~/jboss/jboss-3.0.6/server/uzorg/conf > ls -l
total 236
-rwxr-xr-x    1 josv     users         489 Mar 25 10:40 auth.conf
-rwxr-xr-x    1 josv     users        4562 Mar 25 10:40 jboss-minimal.xml
-rwxr-xr-x    1 josv     users       17232 Mar 25 10:40 jboss-service.xml
-rwxr-xr-x    1 josv     users         306 Mar 25 10:41 jbossmq-state.xml
-rwxr-xr-x    1 josv     users         246 Mar 25 10:40 jndi.properties
-rwxr-xr-x    1 josv     users        6783 Mar 26 17:23 log4j.xml
-rwxr-xr-x    1 josv     users        3862 Mar 25 10:41 login-config.xml
-rw-r--r--    1 josv     users          80 Mar 25 10:41 roles.properties
-rwxr-xr-x    1 josv     users         555 Mar 25 10:40 server.policy
-rwxr-xr-x    1 josv     users       36148 Mar 25 10:40 standardjaws.xml
-rwxr-xr-x    1 josv     users       54893 Mar 25 10:40 standardjboss.xml
-rwxr-xr-x    1 josv     users       76482 Mar 25 10:40 standardjbosscmp-jdbc.xml
-rw-rw-r--    1 josv     users        3647 Apr  7 12:38 upid.properties
-rw-r--r--    1 josv     users          58 Mar 25 10:41 users.properties
```

MBeans hebben overigens niet per definitie een “hot deploy” feature voor de inhoud van configuratiebestanden. Dit is geheel afhankelijk van de implementatie van de MBean die zelfstandig in de gaten moet houden of zijn configuratiebestanden zijn gewijzigd. In sommige gevallen is het nodig om een MBean te undeployen/deployen om wijzigingen in de configuratiebestanden door te laten dringen. Ook kunnen soms MBeans via het JMX Console (zie later in deze paper) worden aangestuurd om hun configuratie opnieuw in te lezen.

Wat soms verwarrend kan zijn is dat sommige instellingen in de MBean XML descriptors in de deploy directory dienen te worden opgegeven, en andere weer in bestanden in de conf directory.

6 JBoss beheer

"Complete adaptation to environment means death.
The essential point in all response is the desire to control environment."

John Dewey
1859-1952, American Philosopher, Educator

6.1 Logging

JBoss maakt voor alle logging gebruik van het Log4J pakket (http://jakarta.apache.org/log4j). Voor applicaties die ook gebruik maken van Log4J geldt dat die logging stroom naadloos wordt opgenomen in de Log4J logging infrastructuur.

De Log4J configuratie file (conf/log4j.xml) specificeert:

- De logging threshold (trace, debug, info, warn, error, fatal) per logging category (normaal gesproken wordt de volledige naam van een klasse gebruikt als logging categorie voor alle meldingen die door die klasse worden gegenereerd). In de Log4J configuratie kan de threshold worden gezet voor een specifieke klasse of een (sub)package.
- Welke logging destinations (appenders) er zijn (inclusief de configuratie van die appenders). Log4J kent een uitgebreide verzameling aan beschikbare appenders (console, file, database, automatische roll-over, etc.). Eventueel kunnen appenders aan het systeem worden toegevoegd (implementatie van het Appender interface) om klant-specifieke logging destinations toe te voegen.
- Welke logmeldingen naar welke appenders moeten worden geschreven. Het is hierbij mogelijk om meldingen naar meerdere appenders te sturen.

Log4J maakt het ook mogelijk om (eventueel per appender) het formaat van de te schrijven logging teksten te bepalen (printf-achtige formaatspecificatie).

De nuvolgende uitsnede van een log4j.xml configuratiefile geeft een impressie van de mogelijkheden:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE log4j:configuration SYSTEM "log4j.dtd">

<log4j:configuration xmlns:log4j="http://jakarta.apache.org/log4j/" debug="false">

  <!-- ================================== -->
  <!-- Preserve messages in a local file -->
  <!-- ================================== -->

  <!-- A time/date based rolling appender -->
  <appender name="FILE"
class="org.jboss.logging.appender.DailyRollingFileAppender">
    <param name="File" value="${jboss.server.home.dir}/log/server.log"/>
    <param name="Append" value="false"/>

    <!-- Rollover at midnight each day -->
    <param name="DatePattern" value="'.'yyyy-MM-dd"/>


    <!-- Rollover at the top of each hour
    <param name="DatePattern" value="'.'yyyy-MM-dd-HH"/>
    -->

    <layout class="org.apache.log4j.PatternLayout">
      <!-- The default pattern: Date Priority [Category] Message\n -->
      <param name="ConversionPattern" value="%d %-5p [%c] %m%n"/>
    </layout>
  </appender>
```

```
  <!-- ============================== -->
  <!-- Append messages to the console -->
  <!-- ============================== -->

  <appender name="CONSOLE" class="org.apache.log4j.ConsoleAppender">
    <param name="Target" value="System.out"/>
    <param name="Threshold" value="INFO"/>

    <layout class="org.apache.log4j.PatternLayout">
      <!-- The default pattern: Date Priority [Category] Message\n -->
      <param name="ConversionPattern" value="%d{ABSOLUTE} %-5p [%c{1}] %m%n"/>
    </layout>
  </appender>

  <!-- Limit JBoss categories to INFO
  <category name="org.jboss">
    <priority value="INFO"/>
  </category>
  -->

  <!-- Decrease the priority threshold for the org.jboss.varia category
  <category name="org.jboss.varia">
    <priority value="DEBUG"/>
  </category>
  -->

  <root>
    <appender-ref ref="CONSOLE"/>
    <appender-ref ref="FILE"/>
  </root>


</log4j:configuration>
```

JBoss (of beter: Log4J) ondersteunt het in-flight wijzigen van de configuratiefile. Na wijziging wordt de nieuwe file (na verloop van tijd) automatisch ingelezen:

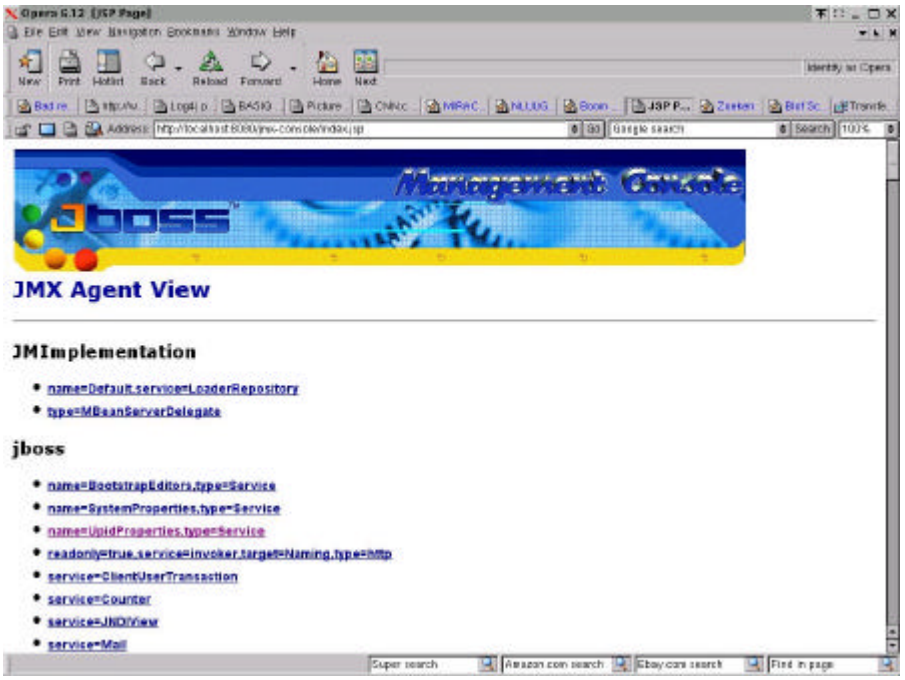
```
08:20:53,358 INFO  [Log4jService$URLWatchTimerTask] Configuring from URL:
resource:log4j.xml
```

Eventueel kan ook via het JMX console (zie volgende sectie) een reconfiguratie van Log4J worden gestart.

6.2 JMX Console

Een van de standaardcomponenten die met JBoss wordt meegeleverd is het JMX Console. Dit is een web applicatie (Java Server Pages) die een rudimentair interface levert op de MBeans die in de JBoss JMX bus zijn geplugd. Via het JMX console kan een overzicht worden verkregen van welke MBeans er actief zijn, wat de waarde zijn van de MBean attributen en kunnen methodes in de MBean worden aangeroepen.

Het JMX Console wordt beschikbaar gesteld via de in JBoss geplugde servlet/JSP engine (Jetty of Tomcat):

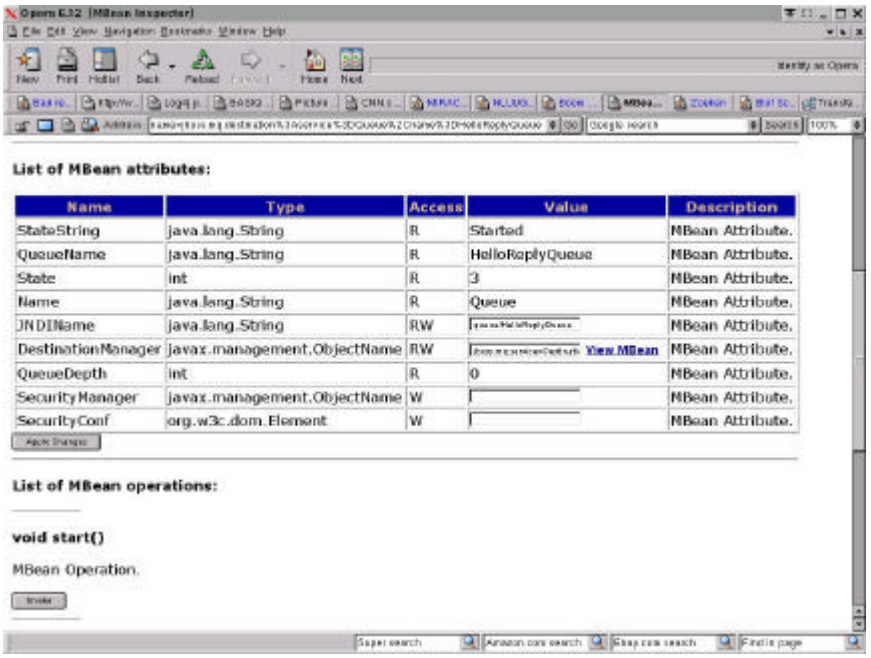


Via het JMX console kunnen allerlei wilde acties worden uitgevoerd (afhankelijk van het precieze type MBean waarop wordt doorgeklikt). De mogelijkheden omvatten onder andere:

- Het aanmaken van nieuwe queues in de JBossMQ subomgeving.
- Het maken van een overzicht van de huidige inhoud van de JNDI server.
- Het aanroepen van een methode in een EJB.
- Het herladen van de Log4J configuratie
- Het wijzigen van MBean attributen
- Het stoppen/starten van een MBean

De JMX console is een zeer algemene faciliteit waarmee in principe alle MBeans (ook van niet JBoss origine, maar in de JBoss JMX bus geplugd) kunnen worden beheerd. De kracht van het JMX console is rechtstreeks gekoppeld aan wat de MBean provider aan attributen en methodes via de JMX API exporteert.

De volgende screenprint laat zien hoe de JMX console er uit ziet voor een bepaalde MBean (in dit geval een JBossMQ queue):



List of MBean attributes:

Name	Type	Access	Value	Description
StateString	java.lang.String	R	Started	MBean Attribute.
QueueName	java.lang.String	R	HelloReplyQueue	MBean Attribute.
State	int	R	3	MBean Attribute.
Name	java.lang.String	R	Queue	MBean Attribute.
JNDIName	java.lang.String	RW		MBean Attribute.
DestinationManager	javax.management.ObjectName	RW	View MBean	MBean Attribute.
QueueDepth	int	R	0	MBean Attribute.
SecurityManager	javax.management.ObjectName	W		MBean Attribute.
SecurityConf	org.w3c.dom.Element	W		MBean Attribute.

List of MBean operations:

void start()

MBean Operation.

6.3 JMX over RMI

JBoss levert ook een RMI (Remote Method Invocation) interface mee op de MBeans in de JMX bus. Via die “RMIAdaptor” kan de JMX bus worden ondervraagd en zijn interacties met de MBeans mogelijk vanuit een Java programma die ergens in het netwerk draait. Op die manier is het mogelijk om softwarematig acties binnenin JBoss te triggeren (b.v. voor de koppeling tussen JBoss en HP OpenView).

7 JBoss voor Java ontwikkelaars

All growth depends upon activity.
There is no development [physically or intellectually] without effort, and effort means work."

Calvin Coolidge
1872-1933, Thirtieth President of the USA

In dit hoofdstuk ga ik in op een aantal JBoss gerelateerde zaken die interessant zijn voor Java ontwikkelaars die applicaties voor JBoss willen bouwen.

7.1 JBoss descriptors

Het is bij de meeste applicatieservers nodig om gedeployde componenten te voorzien van aanvullende (appserver specifieke) deployment informatie. Bij JBoss gebeurt dit volledig door in het componentarchief opgenomen XML descriptors:

jboss.xml	/META-INF in de EJB JAR	Bevat aanvullende EJB eigenschappen zoals de JNDI namen van de beans en de userid/password waarmee Message Driven Beans zich aan de JMS queue connection factory bekend moeten maken.
jbosscmp-jdbc.xml	/META-INF in de EJB JAR	Bevat informatie voor de CMP engine van JBoss (zie verder in dit document) zoals tabel- en kolomnamen, datatypes en pre-load indicatoren.
jbossweb.xml	/WEB-INF in de webapp WAR	Bevat informatie voor het deployen van web applicaties in JBoss zoals JNDI, security en virtual host informatie.

De informatie in de JBoss XML descriptors heeft zowel met de software opbouw als met de operationele karakteristieken van de applicatie te maken. Om die reden is het voor de correcte vulling van die descriptors nodig dat de ontwikkelaars en de operationele beheerders hier afstemming over hebben.

7.2 Container Managed Persistence

JBoss bevat een volledig functionele engine voor ondersteuning van EJB Container Managed Persistence (CMP). CMP is een onderdeel van de EJB specificatie die voorschrijft dat de EJB server zelf de SQL genereert die nodig is om entiteiten in de database te manipuleren. Bij gebruik van CMP hoeft een programmeur alleen maar aan te geven:

1. In welke tabel de entiteit is opgeslagen.
2. Welke velden in de database (naam en type) corresponderen met welke attributen van de EJB.
3. Welke velden foreign keys zijn naar andere tabellen (Container Managed Relations; CMR)

Bij gebruik van CMP is de EJB implementatieklasse abstract, met abstracte getters en setters. Op moment van deploy genereert de appserver dan een concrete implementatie die bij aanroepen van methodes als "ejbCreate", "ejbRemove", "ejbFindXxx", "getXxx" en "setXxx" de benodigde SQL erbij bedenkt en uitvoert. Hierdoor wordt de totale interactie met de database weggehaald bij de programmeur.

Een stukje Java code van een CMP bean ziet er bijvoorbeeld als volgt uit:

```
public abstract class GebruikerBean implements EntityBean {

      …

      /**
       * The CMP get accessor for the Id field.
       */
```

```
      public abstract long getId();

      /**
       * The CMP set accessor for the Id field.
       */
      public abstract void setId(long newId);

      /**
       * The CMP get accessor for the Wachtwoord field.
       */
      public abstract String getWachtwoord();

      /**
       * The CMP set accessor for the Wachtwoord field.
       */
      public abstract void setWachtwoord(String newWachtwoord);

      …
}
```

De JBoss CMP engine is in staat om hier zelf de SQL bij te verzinnen met behulp van de specificaties uit de “jbosscmp-jdbc.xml” descriptor:

```
<entity>
         <ejb-name>Gebruiker</ejb-name>
         <datasource>java:/UpidDS</datasource>
         <datasource-mapping>mySQL</datasource-mapping>
         <create-table>false</create-table>
         <remove-table>false</remove-table>
         <table-name>E002_GEBRUIKER</table-name>

         <cmp-field>
            <field-name>id</field-name>
            <column-name>E002_ID</column-name>
         </cmp-field>

          …
</entity>
```

De “jbosscmp-jdbc.xml” descriptor kan ook allerlei intelligente aansturingen bevatten voor hoe de database te benaderen en welke gegevens gezamenlijk (of vooraf) te laden.

De JBoss CMP engine bevat ook volledige ondersteuning voor het genereren van “finders” op basis van de EJB Query Language (EJB-QL) en het vinden van gerelateerde entiteiten (Container Managed Relations).

7.3 Ontwikkelen voor JBoss

JBoss komt tot ons zonder software ontwikkelomgeving. Het is de appserver, de appserver en niets dan de appserver. In principe kan iedere tool waarmee J2EE applicaties kunnen worden ontwikkeld worden gebruikt voor het schrijven van JBoss componenten. Dit betekent dat zowel de liefhebbers van “vim” en “make” als de verslaafde IDE (Interactive Development Environment) gebruikers voor JBoss kunnen ontwikkelen.

Toch zijn er wel een aantal tools waar ondersteuning voor JBoss in is terug te vinden. Ik wil er hier graag twee noemen: XDoclet en Eclipse.

7.3.1 XDoclet

XDoclet (http://xdoclet.sourceforge.net) is een applicatie waarmee onderdelen van Java applicaties (met name source en descriptor files) kunnen worden gegenereerd. XDoclet doet dit op basis van speciale tags die in commentaarblokken in de Java source code zijn opgenomen. De XDoclet engine verwerkt de gespecificeerde source files, haalt daar de tags uit, correleert die met andere informatie uit de source files en genereert vervolgens (conform instructie) andere files.

XDoclet is met name handig in situaties waarin er een dusdanig nauwe relatie bestaat tussen bepaalde onderdelen van een applicatie dat bijna alle gegevens die nodig zijn voor een componentje uit een andere component zouden kunnen worden verzameld. Dit geldt bijvoorbeeld voor de Home en Remote interfaces van Enterprise Java Beans (die kunnen worden gegenereerd uit de EJB implementatieklasse), alsmede voor de EJB en appserver deployment descriptors (waaronder JBoss).

Daar waar de source files niet genoeg informatie bevatten dient de programmeur XDoclet meta tags te gebruiken om de aanvullende informatie in de source file te specificeren.

Bij gebruik van XDoclet is alle relevante informatie over een EJB in één file opgenomen (de EJB implementatieklasse). Alle gekoppelde informatie wordt gegenereerd. Dit herbergt de belofte in zich van sneller werken en minder fouten. XDoclet werkt als "task" in een Apache Ant build file.

Voor entity beans kan XDoclet onder andere de volgende zaken genereren:

- De vier benodigde Java interfaces (Local, Remote, Local Home en Remote Home).
- Informatie in de ejb-jar.xml EJB deployment descriptor
- Value objects
- Utility objects (bevatten wat handige functies voor het vinden van beans)
- De implementatieklasse voor de primary key van de bean.

Daarnaast kan XDoclet ook de descriptors voor JBoss (“jboss.xml” en “jbosscmp-jdbc.xml”) genereren.

Zoals eerder aangegeven onttrekt XDoclet relevante informatie aan de source code en aan XDoclet meta tags in die source code. Het nuvolgende voorbeeld sluit aan bij het vorige voorbeeld over CMP en bevat de XDoclet tags die nodig zijn om alle relevante sources en (JBoss) descriptors te genereren:

```
/**
 * This class contains the implementation of the Gebruiker
 * entity bean.
 *
 * @ejb.bean                  name="Gebruiker"
 *                            description="Gebruiker"
 *                            jndi-name="ejb/upid/Gebruiker"
 *                            local-jndi-name="ejb/upid/GebruikerLocal"
 *                            type="CMP"
 *                            display-name="Gebruiker"
 *
 * @ejb.value-object          name="Gebruiker" match="*"
 *
 * @ejb.persistence           table-name="E002_GEBRUIKER"
 *
 * @ejb.security-identity     use-caller-identity="true"
 *
 * @ejb.finder    query="SELECT OBJECT(o) FROM Gebruiker o WHERE o.id = ?1"
 *                signature="RemoteGebruiker findById(long id)"
 *                role-name="front-end"
 *
 * @jboss.persistence   datasource="java:/UpidDS"
 *                      datasource-mapping="mySQL"
 *                      create-table="false"
```

```
 *                       remove-table="false"
 */
public abstract class GebruikerBean implements EntityBean {

      …

      /**
       * The CMP get accessor for the Id field.
       *
       * @ejb.interface-method
       * @ejb.permission      role-name="front-end"
       * @ejb.persistence     column-name="E002_ID"
       * @ejb.value-object    match="normal"
       * @ejb.pk-field
       *
       */
       public abstract long getId();

        …
}
```

7.3.2 Eclipse

Eclipse (http://www.eclipse.org) is een “lege” Interactive Development Environment (IDE) die via plug-ins kan worden uitgebreid om een bepaalde programmeertaal of appserver te ondersteunen. Zo zijn er bijvoorbeeld Eclipse plug-ins voor Java, voor C/C++ en COBOL.

Er is momenteel een project gaande (JBoss-IDE) die een JBoss plug-in voor Eclipse aan het bouwen is. Die JBoss plug-in wordt geacht zeer nauw samen te werken met de Java, XDoclet en Ant plug-ins voor Eclipse.

8 JBoss beveiliging

The task we must set for ourselves is not to feel secure,
but to be able to tolerate insecurity.

Erich Fromm
American Psychologist

De beveiliging van een complete JBoss installatie is een complexe aangelegenheid die allerlei niveaus omvat: het besturingssysteem, de JBoss kernel, JBossMQ, EJB declaratieve beveiliging, de beveiliging van (web) applicaties, gebruikerauthenticatie en de database.

8.1 Besturingssysteem

JBoss is in feite een grote Java applicatie die in een JVM draait. Vanuit het besturingssysteem gezien worden dus alle activiteiten (files openen/lezen/schrijven, sockets openen) van alle applicatiecomponenten uitgevoerd met de user waaronder de JBoss JVM draait. Er is in principe geen reden om de JBoss JVM als root/superuser uit te voeren (liever niet zelfs!). De server sockets die JBoss opent gebruiken normaal gespoken poorten die boven de 1024 liggen (dat vereist namelijk root rechten op Unix-achtige systemen).

De beveiliging van de JBoss configuraties en bestanden kan verder geheel via OS features (permissie bits, users/groepen) gebeuren.

8.2 Java Security Manager

In de standaardconfiguraties opereert JBoss zonder Java Security Manager. Door het configureren van de “java.security.manager” property kan een dergelijke security manager in de JBoss JVM worden geactiveerd.

(Voor de niet-Java kenners onder ons: Een Java Security Manager is een object dat allerlei permissievraagstukken door de JVM gedelegeerd krijgt. Zodra een stuk Java code een bestand wil openen, een extern programma uit wil voeren, een netwerkverbinding wil maken of een andere beschermde actie wil uitvoeren dan wordt aan de security manager gevraagd of dit mag. Door de JVM met een security manager te starten kan uitvoerig worden gecontroleerd welke stukken Java code welke acties mogen uitvoeren. De Java Security Manager feature is ook de basis van de veilige “sandbox” waarin applets draaien).

De JBoss implementator heeft de keuze om de standaard Java Security Manager te activeren en daaraan een toegesneden security policy configuratie te hangen of om een eigen security manager te schrijven.

8.3 RMI over SSL

RMI (Remote Method Invocation) is het mechanisme via welke EJB clients verzoeken indienen bij de EJB ’s die onder JBoss draaien. Door het passend configureren van de JBoss client en server is het mogelijk om die RMI verzoeken over SSL verbindingen te sturen. JBoss maakt hiervoor gebruik van de standaard SSL implementatie van Java2: JSSE (Java Secure Socket Extensions). Voor de echte die hards is het vervolgens mogelijk om een eigen RMISocketFactory implementatie te schrijven en daar allerlei aanvullende beveiligingsmaatregelen in te treffen.

Voor andere toepassingen van SSL (b.v. voor HTTP en/of AJP) is JBoss afhankelijk van de mogelijkheden van de geïntegreerde web container (Jetty of Tomcat).

8.4 JAAS

Voor wat betreft de ondersteuning voor applicatieve security maakt JBoss uitgebreid gebruik van de Java Authentication and Authorisation Service: JAAS (standaard onderdeel van Java 1.4; zie ook http://java.sun.com/products/jaas).

JAAS is een pluggable framework voor het op een standaard wijze integreren van een applicatie met allerlei authenticatie en authorisatie diensten waarvan de feitelijke implementatie pas tijdens de uitvoering hoeft te worden geconfigureerd.

JBoss gebruikt JAAS op allerlei mogelijk plekken, maar met name ook voor de authenticatie van EJB clients.

8.4.1 JBoss client login

Voor gebruik door JBoss EJB/RMI clients levert JBoss een “ClientLoginModule” die in de EJB client omgeving in de JAAS configuratie dient te worden gehangen (in de JAAS auth.conf file):

```
// ---------------------------------------------------------------------------
// This is the upid login context used by the test programs. It declares
// that the JBoss client login module is used to perform the
// authentication...
// ---------------------------------------------------------------------------
upid {
        org.jboss.security.ClientLoginModule required;
};
```

Door vanuit de client programmatuur een JAAS login te doen in de “upid” context kan de EJB client zijn JBoss userid en password bekend maken. Die credentials worden niet door de ClientLoginModule geverifieerd maar worden bij het maken van een RMI call naar de JBoss server meegegeven. Dit betekent overigens dat als de credentials incorrect zijn dit pas bij de eerste RMI call blijkt en niet bij de JAAS login!

8.4.2 JAAS JBoss server login

Aan de JBoss server zijde worden de credentials van de client uit de RMI aanroep gehaald en wordt er wederom een JAAS login gedaan, maar dan in de JAAS configuratie van de JBoss server. In de JBoss JAAS configuratie (“conf/login-config.xml’) kunnen dan de feitelijke JAAS login verificatie modules worden opgegeven. Met JBoss komen er een groot aantal modules mee voor verificatie van de user credentials via:

- LDAP
- Database
- Config files

De nuvolgende uitsnede uit de “login-config.xml” beschrijft een standaard login module die werkt op basis van twee configuratiebestanden “users.properties” en “roles.properties” die alle gebruikers, hun wachtwoorden en hun rollen beschrijven:

```
    <!-- The default login configuration used by any security domain that
    does not have a application-policy entry with a matching name
    -->
    <application-policy name = "other">
       <!-- A simple server login module, which can be used when the number
       of users is relatively small. It uses two properties files:
       users.properties, which holds users (key) and their password (value).
       roles.properties, which holds users (key) and a comma-separated list of
       their roles (value).
       The unauthenticatedIdentity property defines the name of the principal
       that will be used when a null username and password are presented as is
       the case for an unuathenticated web client or MDB. If you want to
       allow such users to be authenticated add the property, e.g.,
       unauthenticatedIdentity="nobody"
       -->
       <authentication>
          <login-module code = "org.jboss.security.auth.spi.UsersRolesLoginModule"
             flag = "required" />
       </authentication>
```

```
    </application-policy>
```

Het aldus bepaalde user profiel wordt gebruikt om te bepalen of de EJB client aan de andere kant van de RMI verbinding de aangevraagde methode mag uitvoeren (op basis van de EJB methode rolpermissies die in de EJB descriptor staan vermeld).

8.5 JBossMQ beveiliging

De beveiliging van JBossMQ is nog niet zo hecht geïntegreerd met de rest van JBoss (JAAS). Dit komt onder andere doordat de JMS standaard waar JBossMQ compatibel mee is een andere (veel beperktere) ondersteuning biedt voor het specificeren van user credentials (alleen userid en password bij het verbinden aan de queue manager).

Aan de server zijde is er een aparte JAAS login module (de “DynamicLoginModule”) die users, passwords en rollen geconfigureerd krijgt via een aparte configuratiefile (“conf/jbossmq-state.xml”).

8.6 Database beveiliging

Een van de taken van een J2EE appserver is het op een standaardmanier (CMP) toegang bieden tot de inhoud van databases. In principe gebeurt dit doordat aan een applicatie component (EJB) een datasource wordt gehangen die een pool van connecties naar de database beschrijft. In de JBoss configuratie kan bij de definitie van de pool ook worden opgegeven met welk database userid en password dit dient te geschieden. In JBoss 3.0.6 geschiedt dat door opname van een “ConfiguredIdentityLoginModule” in de JAAS configuratie (waarnaartoe wordt verwezen vanuit de datasource MBean in de deploy directory):

```
<application-policy name = "UpidDbRealm">
       <authentication>
          <login-module code = "org.jboss.resource.security.ConfiguredIdentityLo
ginModule" flag = "required">
             <module-option name = "principal">myuser</module-option>
             <module-option name = "userName">myuser</module-option>
             <module-option name = "password">secretpass</module-option>
             <module-option name = "managedConnectionFactoryName">jboss.jca:serv
ice=LocalTxCM,name=UpidDS</module-option>
          </login-module>
       </authentication>
    </application-policy>
```

In JBoss 3.2+ wordt die informatie in de “*-ds.xml” descriptor gespecificeerd.

Deze wijze van werken heeft tot gevolg dat alle access naar de database onder dezelfde security context plaatsvinden. Het is in principe mogelijk om security context wisselingen te doen in de connecties die uit de datasource worden gehaald, maar beter is het om in de applicatiecomponenten (b.v. door toepassing van EJB declaratieve beveiliging) de applicatieve beveiliging te implementeren.

9 JBoss clustering

"Wishes expand in direct proportion to the resources available for their gratification."

Robert Dato

Een cluster is een groep van nodes die samenwerken om een gemeenschappelijk doel te bereiken. Een JBoss cluster bestaat uit een groep van JBoss instances die samenwerken teneinde fout tolerantie (verhoogde beschikbaarheid) en load balancing (prestatie; performance; schaalbaarheid) te bereiken.

9.1 Features

De JBoss clustering implementatie ondersteunt de volgende features:

- Automatische ontdekking van cluster nodes (zonder additionele configuratie)
- Fail-over en load-balancing features voor:
  - JNDI
  - RMI
  - Entity Beans
  - Stateful Session Beans (met in-memory state replicatie)
  - Stateless Session Beans
- HTTP sessie replicatie met Tomcat en Jetty
- Dynamische JNDI discovery.
  JNDI clients ontdekken automatisch de JNDI InitialContext.
- Cluster-wide gerepliceerde JNDI tree
- Farming
  Gedistribueerde cluster-wide hot-deployment. Hot deploy van een component op één node wordt automatische naar alle andere nodes in de cluster doorgezet.
- Pluggable RMI load-balance policies.
- Een cluster-wide cache invalidation framework (CIF) waarmee cache informatie clusterbreed kan worden geïnvalideerd (JBoss 3.2 en hoger).

De implementatie is gebaseerd op JavaGroups (http://www.javagroups.com), een open source pakket voor multicast communicatie tussen leden van groepen van nodes.

9.2 Client libraries

De JBoss clustering implementatie bevat geen centrale request broker component die verzoeken routeert naar (nog) "levende" nodes van de cluster. Deze switch wordt volledig uitgevoerd door de JBoss clients op basis van ondersteuning voor clustering in de JBoss client libraries (RMI, JNDI) en in de (automatisch) gegenereerde proxy implementaties. Dit betekent dat het voor JBoss client applicaties vrijwel transparant is dat er gebruik wordt gemaakt van een JBoss cluster. Het betekent echter ook dat als er van een niet-JBoss client library gebruik wordt gemaakt (b.v. voor http , AJP of CORBA/IIOP) er een soft- of hardware request dispatcher in de architectuur moet worden opgenomen.

9.3 Cluster configuratie

Een JBoss instance wordt lid van een cluster door de activering van een aantal MBeans die zaken als clustercommunicatie en state replicatie regelen. De standaard "all" server configuratie levert een voorbeeld "cluster-service.xml" mee waarin de benodigde MBeans en hun parameters staan gespecificeerd:

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

<!-- ===================================================================== -->
```

```
<!--                                                                      -->
<!--  Sample Clustering Service Configuration                             -->
<!--                                                                      -->
<!-- ==================================================================== -->

<server>

  <classpath codebase="lib" archives="jbossha.jar"/>

  <!-- ==================================================================== -->
  <!-- Cluster Partition: defines cluster                                   -->
  <!-- ==================================================================== -->

  <mbean code="org.jboss.ha.framework.server.ClusterPartition"
         name="jboss:service=DefaultPartition">

    <!-- Name of the partition being built -->
    <attribute name="PartitionName">DefaultPartition</attribute>
    <!-- Determine if deadlock detection is enabled -->
    <attribute name="DeadlockDetection">False</attribute>
    <!-- The JavaGroups protocol configuration -->
    <attribute name="PartitionConfig">
      <Config>
        <!-- UDP: if you have a multihomed machine,
             set the bind_addr attribute to the appropriate NIC IP address -->
        <!-- UDP: On Windows machines, because of the media sense feature
             being broken with multicast (even after disabling media sense)
             set the loopback attribute to true -->
        <UDP mcast_addr="228.1.2.3" mcast_port="45566"
             ip_ttl="64" ip_mcast="true"
             mcast_send_buf_size="150000" mcast_recv_buf_size="80000"
             ucast_send_buf_size="150000" ucast_recv_buf_size="80000"
             loopback="false" />
        <PING timeout="2000" num_initial_members="3"
              up_thread="true" down_thread="true" />
        <MERGE2 min_interval="5000" max_interval="10000" />
        <FD up_thread="true" down_thread="true" />
        <VERIFY_SUSPECT timeout="1500"
                        up_thread="true" down_thread="true" />
        <pbcast.STABLE desired_avg_gossip="20000"
                       up_thread="true" down_thread="true" />
        <pbcast.NAKACK gc_lag="50" retransmit_timeout="300,600,1200,2400,4800"
                       up_thread="true" down_thread="true" />
        <UNICAST timeout="5000" window_size="100" min_threshold="10"
                 down_thread="true" />
        <FRAG frag_size="8192"
              down_thread="true" up_thread="true" />
        <pbcast.GMS join_timeout="5000" join_retry_timeout="2000"
                    shun="false" print_local_addr="true" />
        <pbcast.STATE_TRANSFER up_thread="true" down_thread="true" />
      </Config>
    </attribute>
  </mbean>

  <!-- ==================================================================== -->
  <!-- HA Session State Service for SFSB                                    -->
  <!-- ==================================================================== -->

  <mbean code="org.jboss.ha.hasessionstate.server.HASessionStateService"
         name="jboss:service=HASessionState">
     <depends>jboss:service=DefaultPartition</depends>
    <!-- Name of the partition to which the service is linked -->
    <attribute name="PartitionName">DefaultPartition</attribute>
    <!-- JNDI name under which the service is bound -->
    <attribute name="JndiName">/HASessionState/Default</attribute>
    <!-- Max delay before cleaning unreclaimed state.
         Defaults to 30*60*1000 => 30 minutes -->
    <attribute name="BeanCleaningDelay">0</attribute>
```

```
  </mbean>

  <!-- ===================================================================== -->
  <!-- HA JNDI                                                               -->
  <!-- ===================================================================== -->

  <mbean code="org.jboss.ha.jndi.HANamingService"
         name="jboss:service=HAJNDI">
     <depends>jboss:service=DefaultPartition</depends>
    <!-- Name of the partition to which the service is linked -->
    <attribute name="PartitionName">DefaultPartition</attribute>
    <!-- RmiPort to be used by the HA-JNDI service
         once bound. 0 => auto. -->
    <attribute name="RmiPort">0</attribute>
    <!-- Port on which the HA-JNDI stub is made available -->
    <attribute name="Port">1100</attribute>
    <!-- Backlog to be used for client-server RMI
         invocations during JNDI queries -->
    <attribute name="Backlog">50</attribute>

    <!-- Client socket factory to be used for client-server
         RMI invocations during JNDI queries -->
    <!--attribute name="ClientSocketFactory">custom</attribute-->
    <!-- Server socket factory to be used for client-server
         RMI invocations during JNDI queries -->
    <!--attribute name="ServerSocketFactory">custom</attribute-->
  </mbean>

  <mbean code="org.jboss.invocation.jrmp.server.JRMPInvokerHA"
         name="jboss:service=invoker,type=jrmpha">
    <!--
    <attribute name="RMIObjectPort">0</attribute>
    <attribute name="RMIClientSocketFactory">custom</attribute>
    <attribute name="RMIServerSocketFactory">custom</attribute>
    <attribute name="RMIServerSocketAddr">custom</attribute>
    -->
  </mbean>

</server>
```

In deze MBean specificatie wordt de parameter “PartitionName” gebruikt om aan te geven van welke cluster deze node lid moet worden. Het is in principe mogelijk om een JBoss instance lid van meerdere clusters gelijk te laten zijn. In dat geval dienen er meerdere cluster configuratie MBeans te worden geactiveerd voor de verschillende partitions.

9.4 HA-JNDI

De JBoss JNDI server is een component via welke toegangsinformatie over geregistreerde JBoss componenten (beans, datasources, queues) kan worden opgehaald. Voor gebruik in een cluster levert JBoss een HA-JNDI implementatie die een gemeenschappelijke JNDI tree onderhoudt voor alle nodes in de cluster.

Bij het deployen van componenten die in clustermode moeten draaien kan dit in de JBoss XML descriptor worden aangegeven:

```
<?xml version="1.0" encoding="UTF-8"?>
<jboss>
 <enterprise-beans>
  <session>
    <ejb-name>MyEJB</ejb-name>
    <jndi-name>MyEJB</jndi-name>
    <clustered>true</clustered>
  </session>
```

```
...

 </enterprise-beans>
</jboss>
```

Één van de effecten van deze instelling is dat de componenten bij de HA-JNDI service worden geregistreerd.

Client applicaties die van de geclusterde componenten gebruik willen maken moeten een lookup voor die componenten doen in de HA-JNDI service. Dit betekent dat ze bij het maken van de te doorzoeken “InitialContext” de toegangsspecificaties (met name het afwijkende poortnummer van HA-JNDI (1100 i.p.v. 1099)) moeten opgeven. Daar het echter een goede gewoonte is om de “InitialContext” properties aan de client applicatie bekend te maken via een externe property file levert dit door de bank genomen geen problemen op. Om in het geval van een node uitval toch JNDI lookups te kunnen bieden ondersteunen de JBoss client libaries de specificatie van een lijst van nodes in de “java.naming.provider.url” property.

Indien een client een zoekactie pleegt voor een object dat niet in de HA-JNDI tree aanwezig is wordt het verzoek doorgestuurd naar alle lokale JNDI implementaties op alle nodes van de cluster. Op die wijze kan een client via HA-JNDI ook een object terugkrijgen wat op een specifieke node leeft.

9.5 Clusteren van EJBs

De JBoss clustering staat het toe om Enterprise Java Beans op meerdere nodes in de cluster te deployen. Verzoeken van client applicaties worden dan dynamisch over de verschillende nodes verdeeld op basis van bean eigenschappen en specifieke configuraties hiervoor in de JBoss XML descriptor van de bean.

9.5.1 Stateless Session Beans

Stateless Session Beans zijn verreweg het meest eenvoudig te clusteren. Doordat deze beans geen state bijhouden van de ene aanroep naar de andere kan iedere aanroep over alle beschikbare nodes en instances worden gescheduled. De “cluster-config” informatie in de JBoss descriptor van de bean kan worden gebruikt om aanvullende informatie hieromtrent te specificeren:

```
<cluster-config>
  <partition-name>DefaultPartition</partition-name>
  <home-load-balance-policy>
    org.jboss.ha.framework.interfaces.RoundRobin
  </home-load-balance-policy>
  <bean-load-balance-policy>
    org.jboss.ha.framework.interfaces.RoundRobin
  </bean-load-balance-policy>
</cluster-config>
```

9.5.2 Stateful Session Beans

Stateful Session Beans (SFSB) zijn wat moeilijker te clusteren aangezien SFSB’s gekoppeld zijn aan een enkele client applicatie instance en intern statii bijhouden van de conversatie met die client. Om een SFSB over meerdere nodes te kunnen clusteren dient de status van de bean na iedere aanroep over de cluster te worden verspreid.

Hiertoe wordt door de standaard “cluster-service.xml” een “HASessionState” MBean geïnstantieerd die verantwoordelijk is voor de state synchronisatie over alle nodes in de cluster. Om dit goed te laten werken is het van belang dat de SFSB zich volledig binnen de EJB specificaties gedraagt en niet stiekem “state” bijhoudt in zaken als (lokale) bestanden (die niet door de state manager worden meegenomen).

Aangezien een JBoss instance in meerdere clusters kan hangen kunnen meerdere HASessionState managers tegelijk actief zijn. In de “cluster-config” entry van de SFSB kan worden aangegeven welke state manager deze bean moet gebruiken.

Verder is het voor een SFSB mogelijk om aan de state manager aan te geven dat er geen gewijzigde state is om te repliceren. Indien de bean een methode bevat met de volgende signatuur:

```
public boolean isModified();
```

wordt die methode door de state manager gebruikt om te achterhalen of de SFSB zijn state gerepliceerd wil hebben.

9.5.3 Entity beans

Entity bean instances representeren entiteiten in de database en worden als zodanig niet geacht andere state bij te houden dan de status in de database. Het clusteren van entity beans is daarom niet problematisch aangezien de database op iedere node beschikbaar is.

Wat echter wel problematisch is (kan zijn) is als entity beans methodes in andere entity beans gaan aanroepen die op andere nodes liggen. De transactionele eigenschappen van beans zouden dan kunnen vereisen dat er een gedistribueerde transactie wordt gestart waarbij de database acties over meerdere nodes in een één transactionele context worden afgehandeld. Momenteel biedt JBoss daar nog geen ondersteuning voor en dient door selectie van de juiste commit optie (in de EJB specificatie) met de “row locking” features van JBoss en de database een oplossing voor dit probleem te worden geconfigureerd. Een uitgebreide behandeling hiervan valt buiten het bestek van deze paper.

9.5.4 Message Driven Beans

Message Driven Beans (MDB’s) worden geactiveerd op basis van het arriveren van een bericht op een queue of topic (publish/subscribe). Doordat JMS van nature al gedistribueerd is hoeven er geen bijzondere maatregelen te worden genomen om de verwerking van dit soort berichten over meerdere nodes te verdelen. Er is echter voor zover ik weet momenteel geen geclusterde implementatie van JbossMQ. Dit betekent (voor zover ik het begrijp tenminste) dat indien de node waarop de queue manager draait uitvalt de gehele queue onbereikbaar is.

9.6 HTTP sessie clustering

Een JBoss cluster ondersteunt ook het clusteren van HTTP sessie informatie zodat requests over meerdere web container implementaties op verschillende nodes in de cluster kunnen worden verwerkt. JBoss levert hiertoe een HTTP session replication service (“jbossha-httpsession.sar”) mee die in de server deploy directory dient te worden geplaatst. Door integratie met Tomcat en Jetty wordt (op basis van de “distributable” vlag van de web applicatie in de “web.xml”) na ieder verwerkt HTTP verzoek het hele HTTP sessie object over de cluster heen gesynchroniseerd. Net als met SFSB sessie replicatie geldt ook hier dat voor de correcte werking hiervan er geen “verborgen” state moet zijn in de vorm van bestanden of andere niet in de HTTP sessie gebonden gegevens.

HTTP sessieobject replicatie zorgt er voor dat iedere node in de JBoss cluster een HTTP verzoek in behandeling kan nemen. De verdeling van verzoeken over de verschillende nodes wordt echter niet door JBoss gedaan maar dient door de HTTP/AJP client te worden gedaan. Mogelijke oplossingen voor die HTTP/AJP client load balancing zijn:

- Uit laten voeren door de web browser (op basis van DNS round robin).
- Een netwerk layer-4 switch (zoals bijvoorbeeld een Linux Virtual Server of een CISCO redirector).
- Voor JBoss een serie Apache web servers hangen en gebruik maken van de load balancing features van mod_jk.

10 JBoss gebruikservaring

“The wise are instructed by reason,
average minds by experience,
the stupid by necessity
and the brute by instinct.”

Marcus T. Cicero
Romeins Orator, Politicus

Op moment van schrijven heb ik JBoss gebruikt in een tweetal projecten (die beide nog niet in productie zijn). Ik wil in dit hoofdstuk graag mijn ervaringen met JBoss op een aantal gebieden in kaart brengen.

10.1 Opstap / complexiteit

De opstap naar het gebruik van JBoss is redelijk hoog. JBoss is een compleet (en dus complex) product en is daarmee vrij lastig in gebruik voor iemand die nog weinig/geen Java en J2EE appserver ervaring heeft. Voor diegenen die daar echter al wel ervaring mee hebben is het een prima product met een heldere, logische, architectuur.

10.2 Installatie / dagelijks beheer

De installatie en het dagelijks beheer van JBoss zijn enorm eenvoudig. Stoppen, starten, herstarten, monitoring en deployment zijn allemaal heel eenvoudig. Met name het JMX Console (eenvoudig, maar functioneel) en het feit dat alle logging via Log4J gebeurt maken het monitoren van wat er aan de hand is relatief eenvoudig.

Het JMX Console kan nog wel wat worden verbeterd, maar dit heeft voornamelijk te maken met de methodes en attributen die door de geïmplementeerde MBeans ter beschikking worden gesteld via het JMX interface.

Aangezien JBoss zelf geen database gebruikt voor het vasthouden van logging of configuratie zijn allerlei manipulaties zoals verplaatsing van directories en dergelijke geen probleem. Een ander voordeel van deze geen-database architectuur is dat er weinig tot geen afhankelijkheden zijn van JBoss naar externe componenten (vind ik zelf nog wel eens een nadeel van WebSphere).

10.3 Build proces

JBoss is een open source product en dus is het mogelijk om de source te downloaden en zelf een (gewijzigde) versie van JBoss te bouwen. Ik heb dat een keer gedaan om meer debugging output te krijgen tijdens het analyseren en oplossen van een probleem rondom het gebruik van de JBoss ClientLoginModule in een multi-threaded omgeving.

De JBoss source tree komt met een Ant build file (“build.xml”) en alle benodigde overige (derde partij) sources en libraries. Het bouwen verloopt probleemloos en levert een distributie tree op die identiek is aan de JBoss binary package.

10.4 Documentatie

JBoss komt met weinig of geen meegeleverde documentatie. De consultants van de JBoss Group schrijven beroepshalve documentatie die (in electronisch formaat) te koop is voor $99,= per jaar (abonnement). Deze documentatie is echter geenszins beginnersdocumentatie te noemen. Er wordt vrij diep op allerlei JBoss internals ingegaan en het heeft tijd (en inzicht) nodig om goed de weg in de documentatie te vinden. Voor de gevorderde JBoss beheerder/programmeur is de documentatie echter een prima informatiebron. Er is echter denk ik behoefte aan documentatie voor de beginnende beheerder en programmeur.

10.5 Community

Net als ieder open source product wordt ook JBoss omringd door een gemeenschap van JBoss ontwikkelaars en gebruikers. Naast de forums op http://www.jboss.org zijn er ook een aantal mailing lists (b.v. "jboss-user@lists.sourceforge.net"). De kwaliteit van de berichtgeving op de mailing list is goed te noemen en ook de kernontwikkelaars van JBoss participeren in de lijst.

10.6 Stabiliteit

Over de stabiliteit van JBoss heb ik tot op heden alleen nog maar goede berichten te melden. Het is me nog niet voorgekomen dat JBoss plotsklaps crashte of ophield te functioneren. Dit wordt mede veroorzaakt door de architectuur en implementatie van Java die veel voorkomende problemen (zoals null-pointer referenties en memory leaks) grotendeels (volledig?) voorkomt.

10.7 Performance

Met de performance van JBoss heb ik nog weinig of geen ervaring alhoewel mijn "gut feeling" me vertelt dat er geen reden is voor grote zorg (gebaseerd op, uuuhhhh, niks). JBoss is zelf een hele grote en complexe Java applicatie en neemt dientengevolge een flinke hoeveelheid geheugen in beslag.