Deployments gebeurden tot nu toe vaak handmatig volgens een gedocumenteerde procedure die precies gevolgd moest worden. Aanpassingen in de procedure die niet of niet goed gedocumenteerd zijn leiden dan al snel tot een problematische deployment, met alle gevolgen van dien. Met een deployment framework zijn deployments te automatiseren en als verlenging van het build proces te gebruiken.

Keep it simple: automatische Java Deployments

Ant, Maven 2 en Continuum

Er zijn genoeg punten waarom deployments van een module vaak niet in een keer goed gaan. De volgorde waarin bepaalde handelingen moeten worden verricht is niet helemaal duidelijk. Externe afhankelijkheden of een kleine aanpassing die handmatig tijdens een deployment wordt gedaan, wordt niet goed gedocumenteerd. Denk aan het goed zetten van de rechten of een bestand aanmaken. Zo zijn er nog meer voorbeelden te bedenken waardoor deployments problematisch kunnen verlopen en slecht reproduceerbaar zijn.

Deployment procedure Om deze problemen aan te pakken moet een deployment allereerst goed gedocumenteerd zijn. De procedure van de deployment moet gescheiden zijn van de deployment zelf. Alle projecten kunnen dan met dezelfde deployment procedure werken, ook al maken ze gebruik van verschillende omgevingen. Dit artikel beschrijft onze oplossing: een framework gemaakt van Ant-scripts. Daarmee kunnen Java-applicaties als package opgeleverd worden uit Maven 2 en automatisch gedeployed worden op een applicatieserver.

Taal & Tools Pareto principe De eerste keuze die je moet maken is welke taal en welke tools je gebruikt. Het uitgangspunt hierbij is dat ze tachtig procent van de gevallen dekken en uit te breiden zijn om de andere twintig procent te dekken. Hierbij is de keuze gemaakt om een generiek stel scripts te maken:

1. voor eenvoudige deployments
   - werken zonder enige aanpassing
   - bijvoorbeeld Tomcat
2. voor complexere deployments
   - uitbreiding mogelijk
   - bijvoorbeeld een WebLogic of WebSphere infrastructuur.

We gebruiken Ant als scripttaal omdat dit platformonafhankelijk is. De deployer en de packager zijn onderdeel van het project en zijn dus beschikbaar voor ontwikkelaars als ze het project uitchecken. Ontwikkelaars kunnen dus zelf een deployment vanaf de command line uitvoeren op een testomgeving.

Veel ontwikkelaars gebruiken Ant als standaardtool voor hun builds en zijn er dus mee bekend. Ant is eenvoudig uit te breiden.We hebben er voor gekozen om Ant-contrib en Ant-jsch als optionele library's toe te voegen om gebruik te kunnen maken van SSH, conditionele logica en Shell scripts.

De Ant-deployment scripts zijn uit te breiden om opdrachten uit te laten voeren door een Shell naar keuze. Dat kan handig zijn als je bijvoorbeeld database scripts of andere scripts wilt laten uit voeren door een Shell en niet door Ant.

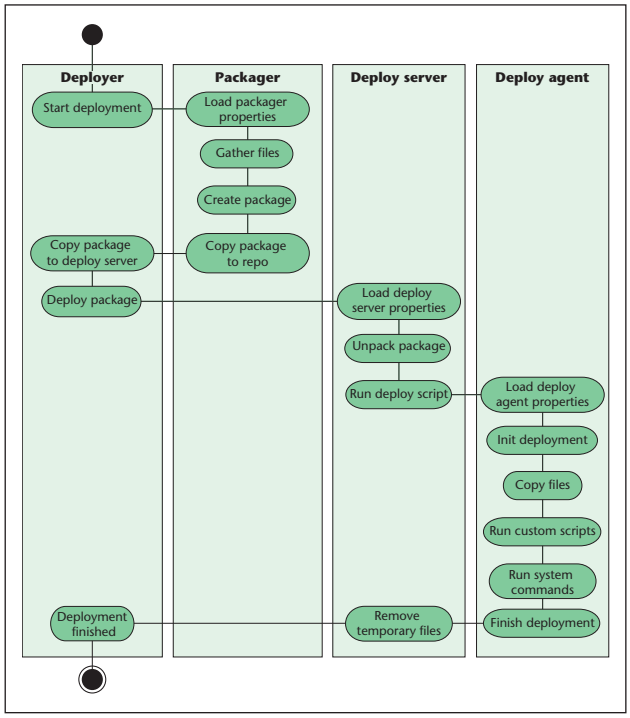
Scripts De deployment scripts worden aan een bepaalde build fase van Maven 2 gekoppeld. Bijvoorbeeld de installfase. Op deze manier wordt de deployment van een module een verlenging wordt van het buildproces. We zijn gemigreerd naar Maven 2 omdat de performance en het geheugengebruik ten opzichte van versie 1 een stuk beter is. De deployer en de packager zijn onderdeel van het project en zijn beschikbaar voor ontwikkelaars als ze het

project uitchecken. Ontwikkelaars kunnen dus zelf een deployment uit voeren op een test omgeving. Deployments kunnen ook automatisch gestart worden met een tool voor build-automatisering, zoals Continuum. Door deployments 's nachts in te plannen of met een lagere frequentie, kunnen ontwikkelaars het resultaat van hun werk snel bekijken op een test omgeving. Dit is wel afhankelijk van de build doorlooptijd en de gewenste vernieuwingsfrequentie van de testomgeving.

SPECIFICATIES De specificaties van een automatische deployment zijn geschreven na een interne discussie. Bij deze discussie zijn verschillende medewerkers betrokken die een brede ervaring hebben met projecten bij onze klanten in alle sectoren en van verschillende omvang. Deze scripts gebruiken wij niet alleen bij al onze interne projecten, maar ook op onze eigen ontwikkel- en test-servers en bij externe hosting providers. Hier maken wij gebruik van Tomcat, MySQL en Linux met een WAR, ontwikkeld met Spring en Hibernate.

Eenvoud zou het uitgangspunt moeten zijn van elke deployment, of elk project eigenlijk (denk Agile), maar in de praktijk heb je toch vaak te maken met complexe omgevingen. Om hier aan tegemoet te komen zijn de scripts modulair en generiek gemaakt. Onze ervaring is dat het aanpassen voor een complexer project dan slechts een kleine extra inspanning kost. De structuur van het framework is als volgt opgebouwd, beginnend in de project root directory:

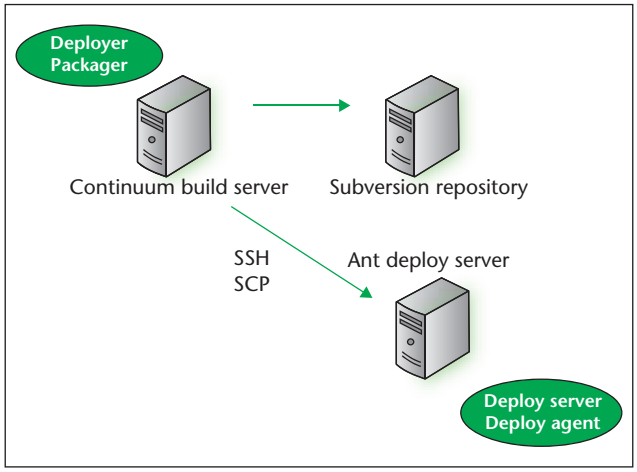
- pom.xml
- **deployer**
  - **deployagent**
    - **conf**
    - **lib**
    - **scripts**
      - init.xml
      - copyfiles.xml
      - runscripts.xml
      - systemcommands.xml
      - finish.xml
    - **static**
    - deployagent.xml
    - deployagent.properties
    - deployagent.properties_<environment1,2...n>
  - deployer.xml
  - deployer.properties
  - deployer.properties_<environment1,2...n>
  - packager.xml
  - packager.properties
  - copytorepo.xml
  - gatherfiles.xml



FIGUUR 1.

De deploy agent heeft een aantal directory's waarin bepaalde typen bestanden in opgeslagen worden die nodig zijn voor een deployment. Met het gatherfiles script kun je de bestanden kopiëren en gebruiken. Voor complexe projecten kunnen de scripts in de scripts directory aangepast worden. Dit zijn aanpassingen zoals het uitvoeren van bepaalde database-scripts, JACL-scripts voor WebSphere, andere custom scripts, of het stoppen en starten van servers.

DE DEPLOYMENT STAPPEN Het deployment-framework kent vier stappen, elk met een eigen script dat volledig generiek is. De packager en de deploy agent roepen een aantal andere scripts aan. De standaard meegeleverde scripts kunnen worden aangepast voor een deployment van een applicatie en werken voor Tomcat en MySQL.



FIGUUR 2.

Het idee achter het framework is dat er voor een deployment verschillende bestanden nodig zijn die worden samengevoegd in een package. Met deze bestanden kunnen de volgende handelingen verricht worden: kopiëren naar een bepaalde locatie, uitvoeren als script, opdrachten geven aan het besturingssysteem, en een deployment initialiseren en afronden. Deze acties vormen de fases in de cyclus van een deployment. Voor elke fase in de deployment cyclus is een uitvoerbaar en aanpasbaar script aanwezig. De deploy agent voert de fases uit op de juiste volgorde.

IMPLEMENTATIE Het project maakt gebruik van de volgende Ant library's:

- Ant-1.6.5
- Ant-jsch
- Ant-contrib
- Ant-run

Ant moet op de deployserver geïnstalleerd worden. Alleen de standaard Ant-1.6.5 library is nodig met Ant-contrib om het deploymentscript te gebruiken. Op de deployserver is een apart deployersaccount nodig met voldoende rechten om een deployment uit te kunnen voeren. Ant-jsch is een optionele library die de deployer nodig heeft. Ant-run wordt door Maven 2 zelf opgehaald en hoeft niet als extra dependency in de POM opgenomen te worden.

Om de scripts te testen en te ontwikkelen gebruiken we in dit voorbeeld Linux en Tomcat. Het proces blijft hetzelfde als van een andere server gebruik wordt gemaakt. Alleen de feitelijke deployment scripts moeten dan worden aangepast. Het werkt het gemakkelijkste als een project een eigen deployersaccount heeft met SSH public/ private keys voor beveiliging, en sudo om opdrachten uit te kunnen voeren. In sudo staan de opdrachten die de deployer mag uit voeren met NOPASSWD zodat deze opdrachten vanuit een Ant script aan te roepen zijn. De rechten van het deployers account kunnen zo beperkt mogelijk zijn.

De sudo configuratie samen met het gebruik van SSH zorgen ervoor dat deployments veilig zijn uit te voeren. Het is dus mogelijk om een applicatie server te hebben waar developers geen toegang toe hebben en waar deployments volledig geautomatiseerd op plaatsvinden. Op een Windows machine speelt beveiliging een kleinere rol. Behalve het installeren van OpenSSH en Ant kunnen de deployment scripts gebruikt worden zonder extra configuratie op het systeem,.

POM.XML In Maven 2 worden de volgende dependencies toegevoegd aan de pom van het project om de optionele Ant tasks beschikbaar te maken in Maven 2:

```
<dependency>
             <groupId>ant</groupId>
             <artifactId>ant-jsch</
artifactId>
             <version>1.6.5</version>
             <type>jar</type>
       </dependency>

  <dependency>
             <groupId>ant-contrib</groupId>
             <artifactId>ant-contrib</
artifactId>
             <version>20020829</version>
             <type>jar</type>
        </dependency>
```

Deze twee optionele Ant-library's zijn te verkrijgen van de Ibiblio Maven 2 repository. De packager wordt aan een build fase van Maven gekoppeld zodat de deployment een uitbreiding vormt van het build-proces:

```
<plugins>
  <plugin>
    <groupId>org.apache.maven.plugins</
groupId>
    <artifactId>maven-antrun-plugin</
artifactId>
    <executions>
      <execution>
        <id>packager</id>
        <phase>deploy</phase>
        <configuration>
        <tasks>
                 <property name="deployment.
environment"
        value="${deployment.environment}" />
                 <property name="deploy.
basedir" value="deployer" />
                 <ant antfile="${basedir}/
${deploy.basedir}/deployer.xml"

   inheritRefs="true" />
        </tasks>
        </configuration>
        <goals>
          <goal>run</goal>
        </goals>
    </execution>
</executions>
</plugin>
```

De scripts zijn zo opgezet dat ze ook afzonderlijk kunnen worden uitgevoerd. Het maken van een package en het uitvoeren van een deployment is met elkaar verbonden, maar dat is niet noodzakelijk. Je kunt er ook

Adv.

voor kiezen om een package op te leveren vanuit Maven 2 of Continuüm, maar dit niet gelijk automatisch te laten deployen.

De package bevat alle scripts en propertiesbestanden om de deployment ook handmatig of via een ander proces uit te voeren op de applicatieserver. De deployserver doet niets anders dan een deployment opstarten en tijdelijke bestanden na afloop weer opruimen. Om het maken van een package te scheiden van het uitvoeren van een deployment, hoef je alleen de property's van het deployerscript te wijzigen.

DEPLOYER.XML Het deployerscript start als eerste de packager en leest vervolgens een property. ${deployment.servers}Hierin staat een komma gescheiden lijst met deployservers, Het deployerscript kopieert met

Elke deployment wordt opgeslagen en is reproduceerbaar voor elke omgeving

gebruik van SCP de package naar elke deployserver. De deployer opent dan een SSH-verbinding en voert het deployment commando uit op elke server in de lijst. Op een eenvoudige wijze is op deze manier ondersteuning voor clustering ingebouwd. Voorwaarde hierbij is dat op elke node een deployer account aanwezig dat identiek is ingericht en de juiste rechten heeft.

Dit werkt bij een infrastructuur waar Tomcat op elke node draait, met een load balancer ervoor. Op een WebSphere server kan dit met een apart deployment script gedaan worden. Als WebSphere goed geconfigureerd is regelt het zelf deployments op een cluster. Dit kan ook met Ant-tasks, maar dan zijn er extra library's nodig.

```
<target name="deployPackage"
description="Deploys package on each
deployment server.">
   <if>

      <equals arg1="${start.deployment}"
arg2="true" />

      <then>

         <typedef name="foreach"
classname="net.sf.antcontrib.logic.ForEach"

   classpathref="maven.runtime.classpath" />

         <!--support for clustering. Make
sure each clustered server has  a deployer

  user account with same username:password-->

         <foreach list="${deploy.servers}"
param="deploy.server" target="copyPackage"

 inheritall="true" inheritrefs="true"
delimiter="," />

         <foreach list="${deploy.servers}"
param="deploy.server"
              target="runDeploymentOnDeploySe
rver" inheritall="true"

            inheritrefs="true" delimiter="," 
/>

      </then>

      <else>

         <echo message="Package created and
not starting a deployment." />

      </else>

   </if>
</target>
```

Deze target laat zien hoe een optionele Ant-task toegevoegd kan worden zodat een Ant-script vanuit Maven 2 gestart kan worden. Hier zie je ook hoe de deployment van de packager losgekoppeld kan worden door de ${start.deployment} property op 'false' te zetten.

PACKAGER.XML De packager heeft twee property's nodig:

- De target omgeving om op te deployen, deze property bepaald welke omgevingsproperty's files er nodig zijn.
- De source directory, waar de deployscripts staan.

De parameter ${basedir} is beschikbaar in Maven. Door het inheritrefs argument op true te zetten wordt het CLASSPATH van Maven beschikbaar in de Ant-scripts. De packager laadt eerst de property's files in:

```
<target name="loadProperties"
description="Load the environment specific
properties file for the packager.">

   <property file="${deploy.basedir}/
packager.properties" />
```

```
   <property file="${deploy.basedir}/pac-
kager.properties_${deployment.environment}"
/>
```

Op deze manier kunnen algemene property's gezet worden die gelden voor alle omgevingen en daarnaast omgevingsspecifieke property's. Dezelfde techniek wordt voor alle property's files gebruikt. De packager haalt de module op. Samen met de bestanden in de deploy directory wordt een zip file gemaakt. Die wordt met een datumtijd stempel en een aanduiding van de omgeving waarnaar gedeployed wordt in een repository gezet. In deze scripts wordt de package in een repository opgeslagen met een datum tijd stempel en kan dus gemakkelijk teruggehaald worden. De packages kunnen in een versiebeheersysteem zoals CVS of Subversion worden opgeslagen, zie het onderstaande voorbeeld:

```
<project name="copyToRepoScript"
default="copyToRepo">
<target name="copyToRepo">

<!--timestamp used for timestamping the
deployment package-->
<tstamp>
   <format property="NOW" pattern=
"yyyyMMd-hhmm" locale="en"/>
</tstamp>

<!--stamped package name-->
<property name="deploy.package.stamped"
       value="${deploy.package}_${deployment.
environment}_${NOW}" />

<copy file="${package.location}/${deploy.
package}"
           tofile="${package.location}/
${deploy.package.stamped}" />

<exec executable="svn">
   <arg value="add" />
   <arg value="${package.location}/${deploy.
package.stamped}" />
</exec>

<exec executable="svn">
   <arg value="commit" />
   <arg value="-m" />
   <arg value="Commited by AutoDeploy" />
   <arg value="${package.location}/${deploy.
package.stamped}" />
</exec>

<delete file="${package.location}/${deploy.
package.stamped}" />
</target>
</project>
```

Elke deployment wordt opgeslagen en is reproduceerbaar voor elke omgeving. Het voordeel hiervan is dat oude releases gemakkelijk kunnen worden teruggezet op de applicatieserver, mits de configuratie van de applicatieservers in de tussentijd niet is gewijzigd.

DeployServer.xml Het deployment script op de deployserver laadt eerst alle property's files en pakt dan het pakket uit in een tijdelijke directory. De deploy scripts uit de package worden als volgt aangeroepen:

```
<target name="runDeployment" description="Run
the deployment script from the unpacked
 package">

<!--read the deployment properties file
delivered with the package-->
        <echo
          message="Reading          ${package.
unpackTo.dir}/${deployment.property.file}_
${deployment.environment}
  as environment property file from the
package."
        />

<property file="${package.unpackTo.dir}/
${deployment.property.file}" />

<property file="${package.unpackTo.dir}/
${deployment.property.file}_${deployment.
environment}"
        />

        <!--run the deployment script
delivered with the package-->
        <ant antfile="${package.unpackTo.dir}/
${deployment.script}" />

</target>
```

Na afloop ruimt het script de package en de tijdelijke bestanden op, zodat op de deployserver geen tijdelijke bestanden achterblijven.

DeployAgent.xml De deploy.xml voert de deployment van de package in vijf stappen uit:

1. init
2. copyfiles
3. runscripts
4. systemcommands
5. finish

Dit is als volgt geïmplementeerd:

```
<project name="deploy" default=
"startDeploymentProcedure" >

<target name="startDeploymentProcedure"
depends="init, copyFiles, runScripts,
         systemCommands, finish" />

  <target name="init" description="This
target is meant to initialize a
 deployment.">

        <echo message="Initializing the
depoyment..." />

        <ant antfile="${readytodeploy.dir}/
${script.dir}/${init.script}" />

  </target>
```

STANDAARDIMPLEMENTATIE De deploy agent kent een deploy cyclus en voor elke stap in de cyclus wordt een script aangeroepen. Niet elke stap in de deploy procedure hoeft gebruikt te worden in ieder project. Voor een eenvoudige deployment volstaat het om in de initialisatie stap de server te stoppen, de webmodule te kopiëren, een databasescript te draaien in de runscripts stap, de module uit te pakken in de systemcommands stap, en in de finish stap de server weer te starten. Deze standaarddeployment kan met een kleine inspanning van een paar uur uitgerold naar een nieuw project. De enige aanpassingen zijn wijzigingen in property's files, zoals directory paden en hostnames.

In de deploymentprocedure zijn geen wijzigingen nodig. Voor een nieuwe release kan wel een verandering in de deployment nodig zijn. Dit betreffen dan wijzigingen zoals een database-conversie script dat moet worden uitgevoerd. Hiervoor zijn alleen aanpassingen nodig in de scripts die door de packager of de deploy agent worden aangeroepen.

UITBREIDINGEN Voor ons testproject hebben we een implementatie geschreven met Subversion en Tomcat die opnieuw bruikbaar is. Deze verzameling scripts vormen een deploymentframework dat uit te breiden is voor complexere projecten. Het is mogelijk om custom scripts toe te voegen in andere scriptingta-

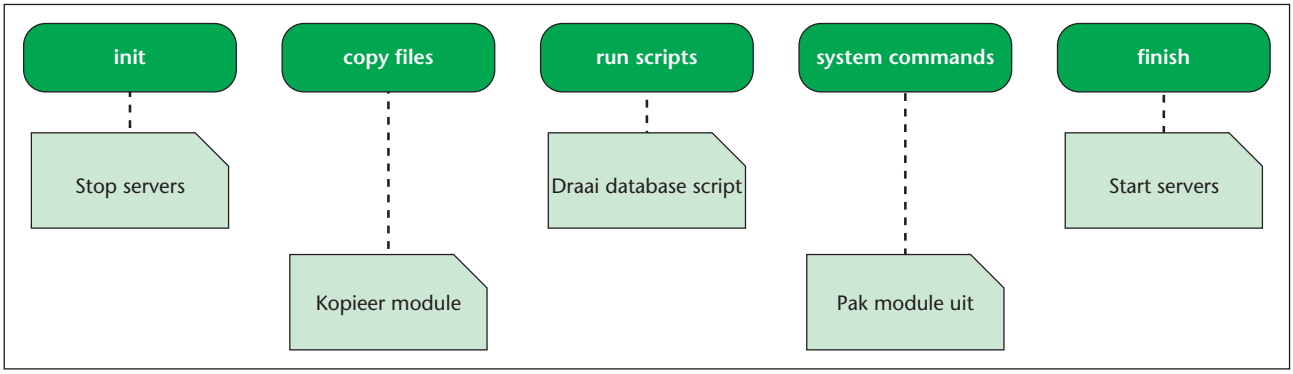
De deploy agent kent een deploy cyclus; voor elke stap in de cyclus wordt een script aangeroepen

len, zoals JACL of PL/SQL. Deze roep je dan aan in je runscripts.xml. De eenvoudigste implementatie ziet er als volgt uit:

```
<typedef name="shellscript" classname="net.
sf.antcontrib.platform.ShellScriptTask"/>

<shellscript shell="${shell}"
dir="${readytodeploy.dir}/scripts"
       inputstring="${runscripts.command}" />
```

Je hoeft dus alleen per fase in de deploy cyclus een property te wijzigen per omgeving waarop je gaat deployen. In dit geval: ${runscripts.command. Eenvoudige handelingen kun je op deze manier toevoegen door de property's aan te passen. Als je iets ingewikkelder wilt met meer handelingen zul je het runscripts.xml bestand zelf moeten aanpassen om bijvoorbeeld een Shell-script uit te laten voeren dat bestaat uit meerdere regels met conditionele logica erin. Dit druist echter in tegen het uitgangspunt van het framework, eenvoud. Om JACL- of PL/SQL-scripts uit te laten voeren kun je denken aan een oplossing waarin je runscripts.xml aanpast met extra statements om de JACL- en PL/SQL-scripts aan te roepen. Herstarten van Tomcat is als volgt geïmplementeerd:



FIGUUR 3.

```
<project name="SystemCommands" default="syste
mCommands">

  <target name="systemCommands">

     <exec command="sudo /etc/init.d/tomcat
stop" />

     <exec command="sudo /etc/init.d/tomcat
start" />
```

Conclusie Eenvoud en reproduceerbaarheid zijn de kenmerkende uitgangspunten van dit framework. Door dit framework te implementeren en aan een Maven buildfase te binden is een deployment een vast onderdeel van Continuous Integration.

Voordelen hiervan zijn dat je in je projecten de unit tests kunt automatiseren met bijvoorbeeld Continuum, en nu dus ook de integratietests. Ontwikkelaars kunnen zelf vanaf de command line een deployment uitvoeren op een test omgeving. Dit framework kan worden geïnstalleerd op de productieomgeving zodat deployments controleerbaar en reproduceerbaar uit te voeren zijn. Systeembeheerders hebben geen specifieke kennis nodig van Java deployments. Deployments zijn gemakkelijker terug te rollen.

De tijd die nodig is om deployments uit te voeren op test-, acceptatie- en productieomgevingen wordt verkort. Het is van belang om een goede simulatie van de productieomgeving te maken. Alleen dan heeft een automatische deployment als onderdeel van de Continuous Integration zin.

Dit framework is flexibel en schaalbaar. Het is modulair opgezet en gemakkelijk uit te breiden voor verschillende omgevingen. Het is een zinvolle en nuttige uitbreiding op tools zoals Continuum en Maven. Het framework is als open source te downloaden van: http://www.iprofs.nl/article/.

Reinout Korbee is werkzaam bij iProfs als J2EE-specialist.