Vraag een Javaan naar de gebruikte tools in zijn ontwikkelstraat en grote kans dat Maven als buildtool wordt genoemd. Bij de release van Maven 3 in oktober 2010 zijn nieuwe features geïntroduceerd en is refactoring toegepast. Ook voor Maven 3.1 staan verbeteringen te wachten. Is Maven 3 dan de tool voor build management of zijn er inmiddels betere tools beschikbaar?

Maven 3 tegenover Gradle en Buildr

Vergelijking van build management tools

Thijs Volders
Senior JavaEE/SOA Consultant & Architect bij Yenlo

Maven is een build management tool waarmee, onafhankelijk van de gebruikte IDE, software gebouwd, gepackaged en gedeployed kan worden. Maven is sinds versie 2 een veelgebruikte tool in menig ontwikkelstraat en heeft de laatste jaren zijn weg gevonden in vele build-gerelateerde tools. Maven hanteert een strikt build proces, gebruikt een projectconfiguratie die is opgebouwd uit XML en biedt mogelijkheden om plugins toe te voegen. In Maven 3 zijn onder de motorkap veel functionaliteiten verbeterd. Merk je hier echter bij het gebruik veel van? Veel gebruikers kennen de Maven 2 functionaliteiten goed en moeten de Maven3 functionaliteiten nog ontdekken. Ondanks de vele interne wijzigingen kan Maven 3 zonder conversies Maven 2 projecten bouwen.

We nemenl Maven 3 als referentie tool met daarnaast Gradle en Buildr als alternatieven. De eerste major release van Maven was in 2005 en inmiddels is Apache Maven in zijn derde major release belandt met een grote gebruikersbasis.
Gradleware Gradle is het jongste alternatief. Gradle bevindt zich nog in een prille fase. Het project is eind 2009 gestart en de 1.0 milestone release is volgens hun JIRA systeem gereleased op 27 februari 2011. De tool Gradle wordt gesupport door GradleWare. Zij leveren zowel support als training voor deze tool.

Buildr is eveneens een Apache gehost project. Het bestaat sinds eind 2007. De meest recente release is 1.4.5 en deze is gereleased op 20 februari 2011. Wat zijn de belangrijkste verschillen tussen deze drie tools? Zijn Gradle en Buildr waardige vervangers of wellicht (toekomstige) opvolgers voor Maven 3? Om dit te bepalen doorlopen we een aantal features van deze tools.

Projectconfiguratie

Voor velen is de Maven pom.xml bekend maar ter illustratie hieronder een voorbeeld van een simpel project:

```
<project xmlns=http://maven.apache.org/POM/4.0.0
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
         http://maven.apache.org/xsd/
maven-4.0.0.xsd"><modelVersion>4.0.0</modelVersion>
<groupId>nl.yenlo.voorbeeld</groupId>
<artifactId>maven3-app</artifactId>
<version>1.0-SNAPSHOT</version>
<packaging>jar</packaging>
<dependencies>
         <dependency>
                   <groupId>org.testng</groupId>
                   <artifactId>testng</artifactId>
                   <version>[5.14.1,)</version>
                   <scope>test</scope>
         </dependency>
</dependencies>

<build>
   <plugins>
      <plugin>
         <groupId>org.apache.maven.plugins</groupId>
<artifactId>maven-compiler-plugin</artifactId>
<configuration>
         <source>1.6</source>
         <target>1.6</target>
   </configuration>
  </plugin>
</plugins>
</build>
</project>
```

Maven hanteert projectconfiguraties in de vorm van pom.xml files en verwacht een project- overstijgende configuratie in de vorm van settings.xml. Deze settings.xml bevat onder andere specifieke gebrui-

kerinstellingen zoals credentials voor het benaderen en installeren van libraries op repositories.

Sinds de komst van Maven 3 is het mogelijk geworden om met behulp van Polyglot Maven de projectdefinitie in een andere taal dan XML te schrijven. Polyglot Maven is geen onderdeel van de Maven 3 core en is nog volop in ontwikkeling. Het lijkt perspectieven te bieden om de projectconfiguratie van Maven op een flexibelere manier te definiëren.
Maven-projecten doorlopen een aantal standaardfases binnen lifecycles. Het Maven build model is hier helemaal op gebaseerd. Door plugins aan fases binnen lifecycles te 'binden' wordt het totale buildproces doorlopen.
Om bovenstaand project om te bouwen en te packagen is 'mvn clean package' voldoende. Hierdoor worden de lifecycles en fases doorlopen en wordt een jar-file gecreëerd. Maven is voornamelijk gericht op de buildmanagement van Java-projecten. De convention- over -configuration tactiek laat dit duidelijk zien. Veel configuratie van Maven heeft standaardinstellingen die op Java-projecten van toepassing zijn.

Gradle definieert projectconfiguratie met behulp van een build.gradle file. De inhoud van deze file is, in tegenstelling tot Maven, een op Groovy gebaseerd buildscript. De configuratie is compact en bevat weinig boilerplate code. Een voorbeeld van een Gradle projectdefinitie:

```
apply plugin: 'java'
sourceCompatibility: 1.6

repositories {
         mavenCentral()
}
dependencies {
         testCompile group: 'org.testng', name:
'testng', version: '5.14+'
}

project_name="gradle-app"
group='nl.yenlo.voorbeeld'
version='1.0'
```

Een Gradle buildfile bestaat uit een aantal tasks. Deze tasks kunnen zelf worden gedefinieerd of met behulp van een plugin beschikbaar worden gemaakt binnen deze configuratie. In bovenstaand voorbeeld wordt door de Java-plugin een aantal tasks geïntroduceerd die voor Java-projecten van toepassing zijn. Out of the box doorloopt Gradle geen standaard lifecycle/proces. De gebruiker definieert zelf de tasks, die moeten worden doorlopen, waarbij een task of groep van tasks het totale build-proces definiëren. Door een plugin toe te voegen aan de buildfile kunnen tasks beschikbaar worden gesteld binnen het build-proces.

De Java-plugin in bovenstaand voorbeeld voegt een aantal tasks toe die overeenkomen met lifecycles zoals deze bekend in zijn Maven. Gradle is in tegenstelling tot Maven meer dan een Java buildmanagement tool. Gradle ondersteunt naast de Java-plugin ook andere plugins, waarmee op andere platform gebaseerde projecten gebouwd kunnen worden. Bij Gradle is het build-by-convention principe van toepassing. Het verschil met Maven's convention-over-configuration is dat Maven zelf een strikt build-proces volgt, waar bij Gradle een (collectie van) plugin(s) de 'convention' definieert. Gradle definieert heel beperkt een proces. Iedere plugin kan bepaalde tasks beschikbaar stellen die samen een resultaat opleveren. Bij een leeg Gradle-project gebeurt er niets als er een Gradle install commando wordt gegeven. Door de Java-plugin toe te passen wordt de sourcecode opgezocht op bepaalde standaardlocaties en wordt deze gecompileerd, getest etceetra.

Buildr projectconfiguratie wordt gedefinieerd in een buildfile. Deze buildfile is op Ruby gebaseerd. Een buildfile voor het bovenstaande Maven project:

```
repositories.remote << 'http://www.ibiblio.org/maven2'
define 'buildr-app' do
         project.version='1.0'
         project.group='nl.yenlo.voorbeeld'
         package :jar
         test.using :testng
         compile.using :source=>'1.6', :target=>'1.6'
end
```

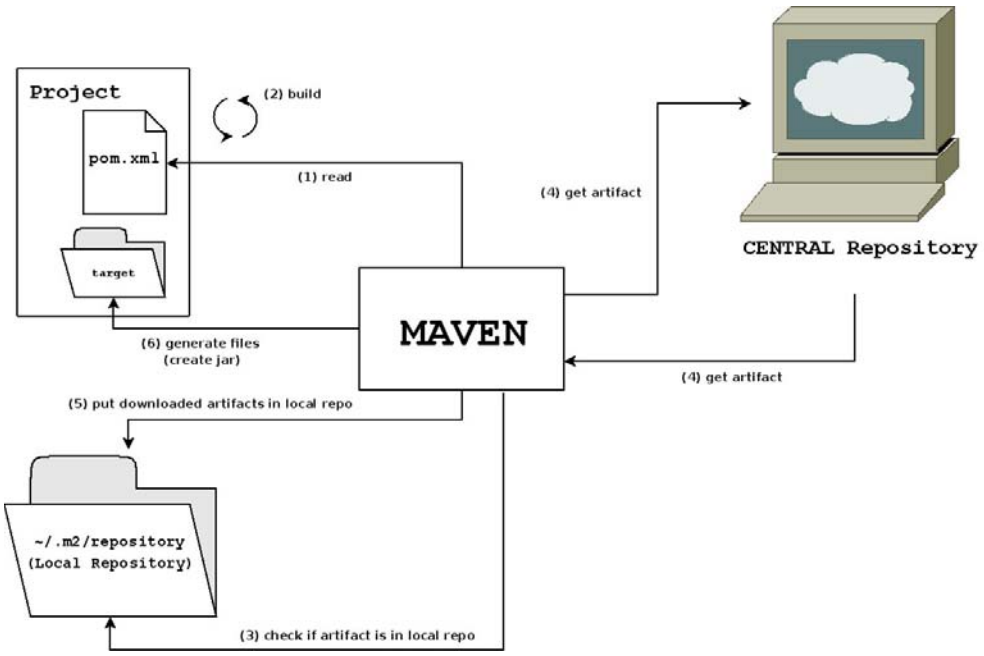
Ook hier valt direct op dat de buildfile erg compact is in tegenstelling tot Maven. Buildr definieert een aantal standaardvariabelen die binnen de buildfile van een waarde voorzien kunnen worden. Door net als Gradle een flexibele taal als basis te gebruiken heeft ook Buildr vergaande flexibiliteit in het definiëren van het build proces.

Net als Maven en Gradle heeft ook Buildr mogelijkheden om uitbreidingen aan het build script toe te voegen. Binnen Buildr zijn dit in feite Ruby functies. Buildr kent standaard de build-task. Deze task is in staat om een aantal standaard directorystructuren te bekijken en op basis van het aanwezig zijn bestanden in deze directories het project te builden. Als bijvoorbeeld java-sources in src/main/java staan dan zal het project als een java project worden gecompileerd. Als er files staan in src/main/scala dan wordt het geheel als scala project behandeld. Ook hier komt duidelijk het build-by-convention principe naar voren.

Artifacts en Repositories

Maven gebruikt een artifact repository voor het opslaan en managen van third-party libraries. De meerwaarde van een dergelijke repository is niet onopgemerkt gebleven en is door vele tools in gebruik genomen. Zo ook door Gradle en Buildr. Beide tools (en natuurlijk Maven zelf)

Net als Maven en Gradle heeft ook Buildr mogelijkheden om uitbreidingen aan het build script toe te voegen.



Hoe Maven werkt.

Plugins schrijven voor Maven vraagt ook in versie 3 veel programmeerwerk.

kunnen gebruik maken van een dergelijke artifact repository.
Gradle biedt beperkte additionele mogelijkheden in het gebruiken van artefacten. Het is mogelijk om (net als in Maven) één of meerdere repositories toe te voegen aan de buildfile. Het is configurabel of deze repositories de Maven standaard volgen. Indien dit niet het geval is, dan kan een aangepaste folder-structuur worden gedefinieerd waarbinnen de artifacts gevonden kunnen worden.

Buildr heeft geavanceerdere mogelijkheden voor het herbruiken van artifacts. Naast de support voor de Maven repositories is Buildr ook in staat om artifacts die niet de Maven standaard volgen te gebruiken. Ter illustratie:

```
DOJO = '0.2.2'
url = "http://download.dojotoolkit.org/release-#{DOJO}/
dojo-#{DOJO}-widget.zip" download(artifact("dojo:dojo:
zip:widget:#{DOJO}")=>url)
```

Deze buildregels zorgen ervoor dat de dojo-0.2.2-widget.zip gedownload wordt van de dojo- website en wordt opgenomen in het build-script als dependency. Ook kan Buildr een artifact als ZIP downloaden, de Jar uitpakken en vervolgens automatisch in de lokale repository installeren.

Sinds Maven 3 is de ondersteuning voor aangepaste repository layouts verbeterd, echter is hier in de documentatie weinig informatie over te vinden. Dit strookt ook met de filosofie van Maven om zoveel mogelijk aan de Maven definities vast te houden en zo min mogelijk aanpassingen toe te passen.

Plugins

Plugins voor Maven schrijven is ingewikkeld en omvat veel programmeerwerk. Helaas is dit in Maven 3 niet veel verbeterd. De API is op bepaalde punten wel aangepast, maar dit heeft weinig effect op de hoeveelheid boilerplate code die nodig is voor een plugin. In Maven 3 is het wel mogelijk geworden om plugins op een makkelijkere manier uit te breiden door extension points te gebruiken.

In Gradle en Buildr daarentegen is het opzetten van een plugin relatief weinig werk. In Gradle kan een plugin direct in de build.gradle file worden geschreven. Ook kan een externe implementatie worden gemaakt en is het mogelijk om deze dan beschikbaar te stellen binnen het build-script. Een task kan eveneens direct in de build.gradle file worden geschreven of daarbuiten beschikbaar worden gemaakt. Gradle ondersteunt het schrijven van plugins en tasks in Groovy, Scala of Java. In feite is iedere taal mogelijk zolang het resulteert in gecompileerde class files. Gradle compileert automatisch build-scripts en stelt deze beschikbaar binnen het project tijdens het build-proces.

Ook Buildr ondersteunt aanpassing van het build-proces in een relatief eenvoudige vorm. Buildr ondersteunt het schrijven van plugins in Ruby/Rake. Tasks kunnen in rake files worden gedefinieerd en deze kunnen op verschillende manieren beschikbaar gesteld worden tijdens het build-proces. Om dergelijke tasks te delen over meerdere projecten kunnen ze bijvoorbeeld als Ruby Gems gedefinieerd worden. Maar het is ook mogelijk om bij het project zelf op te slaan.

Flexibiliteit

Zoals al eerder aangegeven is Maven (ook in versie 3) strikt als het gaat om het buildproces. Er is een precieze definitie van de fases die worden doorlopen om een buildproces uit te voeren en plugins kunnen aan de verschillende fases van het proces worden gekoppeld. Er zijn beperkte mogelijkheden om dit proces te beïnvloeden. Door bepaalde lifecycles op te geven op de command-line kunnen stappen van het proces worden uitgesloten of overgeslagen. Gradle en Buildr daarentegen definiëren een vele malen minder strikt buildproces. Buildr definieert out-of-the-box een standaard Java-project build task en Gradle definieert een leeg project. Zowel Gradle als Buildr bieden mogelijkheden om het build-proces volledig naar wens in te richten en geven ook mogelijkheden om met behulp van plugins complexe en geavanceerde acties uit te voeren tijdens het buildproces.

Eén van de voordelen van Maven als vervanger van Ant was het duidelijke buildproces. Een Ant build file bood veel meer flexibiliteit, maar Maven bood daarentegen een vele male simpelere opzet van een project. Dit is ook een van de nadelen aan Gradle en Buildr. Het gebruik van deze tools zorgt ervoor dat ieder project potentieel een heel ander buildproces definieert, waardoor aanpassingen binnen een complex buildproces lastig kunnen blijken.
Eén van de nadelen van Maven is dat projectconfiguraties delen alleen via overerving kan gaan. Een project definieert een bepaalde parent waardoor configuraties beschikbaar komen binnen de projectpom. Gradle en Buildr lossen dit probleem op door met behulp van plugins en tasks dergelijke configuraties op te delen in herbruikbare componenten.

Voor Maven 3.1 staat 'Mixins' gepland. Dit zal Maven de mogelijkheid geven om geparameteriseerde POM-componenten binnen ander POMs te herbruiken. Behalve dat deze functionaliteit gepland staat is verder erg weinig bekend over de precieze uitwerking.

Wie mist wat?

In voorgaande paragrafen zijn voornamelijk de verschillen van soortgelijke functionaliteiten vergeleken, maar er zijn ook functionaliteiten behandeld die wel door de ene, maar niet door de andere tool worden geboden.
Maven 3 biedt een shell omgeving. Deze shell start de JVM, doet de artifact resolving en houdt artifacts en plugins in geheugen. Dit bespaart bij vervolgbuilds aanzienlijk tijd. Verschillende tests laten tientallen procenten tijdswinst zien bij zowel simpele als complexe projecten. Developers die veelal projecten builden vanaf de command-line zullen deze shell kunnen gebruiken.
Buildr biedt een mogelijkheid om als daemon te worden opgestart. Buildr blijft dan op een vooringestelde tijdsinterval de sourcecode tree pollen om gewijzigde files vervolgens opnieuw te compileren. Bij iedere tijdsinterval wordt de compile task opnieuw uitgevoerd. Unit tests worden tijdens dit proces niet uitgevoerd, evenals wijzigingen in dependencies die worden gedaan in het buildscript. Deze deamon modus wordt tegenwoordig veelal door de IDE geregeld, hoewel niet iedere IDE dit even goed afhandelt. Door deze functionaliteit kan dit mogelijk opgelost worden.

Gradle biedt de mogelijkheid om de Gradle executable met het project mee te leveren. Dit kan door het toevoegen van een wrapper task binnen de buildfile worden geregeld;

```
task wrapper(type: Wrapper) {
         gradleVersion = '1.0-milestone-1''
}
```

Als deze task wordt uitgevoerd wordt de aangegeven versie van Gradle gedownload en bij het project gevoegd. Dit resulteert in de Gradle jar, een shell script en een windows batch file. Deze files kunnen vervolgens gebruikt worden om op een systeem, waar de user geen rechten heeft om nieuwe software te installeren, het project te bouwen. Dit kan bijvoorbeeld een Continous integration server zijn.

Conclusie

In dit artikel zijn drie build management tools met elkaar vergeleken te weten Maven v3, Gradle v1.4.x en Buildr v1.0. Maven heeft duidelijk als voordeel dat het zo'n grote gebruikerscommunity heeft. Er zijn vele plugins beschikbaar die uiteenlopende functionaliteiten bieden om verschillende aspecten van een project build te faciliteren.

Het strikte buildproces wordt veelal als een obstakel gezien en ook Maven 3 wijkt hier nauwelijks van af. Ook wordt de XML project configuratie als een probleem ervaren. Het is duidelijk dat tools als Gradle en Buildr, die een flexibelere taal hanteren, meer aanpassingen aan het buildproces toelaten. Gradle en Buildr bevatten allebei veel minder plugins om de build te verrijken met additionele informatie. Dat werkt op dit moment mijns inziens tegen ze. Beide alternatieven hebben potentie om als Maven vervangers te dienen, maar wederom gezien het beperkte aantal build-proces verrijkende tools alsook de beperkte ondersteuning van Gradle en Buildr in IDE's, Continous integration servers e.d. lijkt dit nog een stap te ver. Gaat de komst van Polyglot Maven de alternatieven weer buiten spel zetten? Maven krijgt dan immers ook meer flexibiliteit in het definiëren van een projectconfiguratie. Het is wachten op de toekomst.... «

Maven heeft het voordeel van een grote gebruikers community en er zijn veel plugins beschikbaar.