

Gant as Ant and Maven Replacement

Dr Russel Winder
Concertant LLP

russel.winder@concertant.com
russel@russel.org.uk



Groovy and Grails User Group

Aims and Objectives

Convince people that XML is a poor way of representing build information and that a Groovy script is far better.

Introduce Gant as a lightweight framework to make describing builds in Groovy scripts even easier.

Convince people that Groovy is Grrrrrrrrrrrrreat.

Foment (or should that ferment 😊) debate.



Problem 1 – XML

XML looks ugly and is hard to read and write.

Is this a problem?

Ant – definitely.

Maven – possibly.

Maven here always means Maven 2.

Maven 1 is ignored as being anathema.

Maven 1 uses Jelly, quod erat demonstrandum.



An Example Complex Project

Hello World is a suitable candidate.

```
public class HelloWorld {  
    public static void main ( final String[] args ) {  
        System.out.println ( "Hello World." );  
    }  
}
```



Ant

```

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

<project name="HelloWorld" default="test" basedir=".">

  <property name="sourceDirectory" value="Source"/>
  <property name="buildDirectory" value="Build"/>

  <target name="compile" description="Compile sources.">
    <mkdir dir="${buildDirectory}"/>
    <javac srcdir="${sourceDirectory}" destdir="${buildDirectory}" debug="on"/>
  </target>

  <target name="test" depends="compile" description="Compile and then run tests.">
    <java classname="HelloWorld" classpath="${buildDirectory}"/>
  </target>

  <target name="clean" description="Clean the detritus.">
    <delete dir="${buildDirectory}" quiet="true"/>
    <delete quiet="true">
      <fileset dir="." includes="**/*~" defaultexcludes="no"/>
    </delete>
  </target>
</project>

```

```

.
|-- Build
|   |-- HelloWorld.class
|-- Source
|   |-- HelloWorld.java -> ../../HelloWorld.java
-- build.xml

```



Maven (Part 1 of 4)

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

<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/maven-v4_0_0.xsd"
  >
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.concertant.examples</groupId>
  <artifactId>helloWorld</artifactId>
  <packaging>jar</packaging>
  <version>0.1.0</version>
  <name>The Hello World Program</name>
  <description>
    An example of Maven POM.
  </description>
  <url>http://www.concertant.com</url>
  <inceptionYear>2007</inceptionYear>
  <licenses>
    <license>
      <name>GPL</name>
      <url>http://www.gnu.org/licenses/gpl.html</url>
    </license>
  </licenses>
</project>
```



Maven (Part 2 of 4)

```
<developers>
  <developer>
    <id>russel</id>
    <name>Russel Winder</name>
    <email>russel.winder@concertant.com</email>
    <organization>Concertant LLP</organization>
  </developer>
</developers>

<build>
  <defaultGoal>test</defaultGoal>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-compiler-plugin</artifactId>
      <configuration>
        <source>1.6</source>
        <target>1.6</target>
        <debug>on</debug>
      </configuration>
    </plugin>
```



Maven (Part 3 of 4)

```
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-antrun-plugin</artifactId>
  <executions>
    <execution>
      <id>test</id>
      <phase>test</phase>
      <configuration>
        <tasks>
          <java classname="HelloWorld" classpath="target/classes"/>
        </tasks>
      </configuration>
      <goals>
        <goal>run</goal>
      </goals>
    </execution>
  </executions>
</plugin>
```



Maven (Part 4 of 4)

```
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-clean-plugin</artifactId>
  <configuration>
    <filesets>
      <fileset>
        <directory>.</directory>
        <includes>
          <include>**/*~</include>
        </includes>
        <followSymlinks>>false</followSymlinks>
      </fileset>
    </filesets>
  </configuration>
</plugin>
</plugins>
</build>
</project>
```



Gant

```
sourceDirectory = 'Source'
buildDirectory = 'Build'

target ( compile : 'Compile sources.' ) {
  Ant.mkdir ( dir : buildDirectory )
  Ant.javac ( srcdir : sourceDirectory , destdir : buildDirectory , debug : 'on' )
}

target ( test : 'Compile and then run.' ) {
  depends ( compile )
  Ant.java ( classname : 'HelloWorld' , classpath : buildDirectory )
}

includeTargets << gant.targets.Clean
cleanDirectory << buildDirectory
cleanPattern << '**/*~'

target ( 'default' : 'Default is test.' ) { test ( ) }
```

Builders Rock!

DSLs Rule.

Groovy is the
bees knees.



Problem 2 – XML

XML is fine for representing data.

Is a build specification just data?

Ant – no.

Maven – possibly.



Ant – The Problem

All too often:

Targets have to be used as procedures.

Guarded targets get used as if statement.

Clearly there is a problem:

Defensiveness of Ant people about “being declarative”.

Plugins all have to deal with dependency management.

The existence of additions for providing control flow structures.



Maven – Not such a Problem

Maven has structuring and naming rules that mean process can be hidden behind declarative specification of state.

Plugins do all the work – if there is a plugin to do what you want then fine, otherwise . . .

If your build doesn't fit the Maven model then you are deep in the effluent.



Ant → *Gant*

Replacing Ant with Gant seems a “no brainer”:

Gant uses all the Ant tasks: Gant uses Ant infrastructure, it does not replace – though it does augment.

The XML driver is replaced with a Groovy script:

Declarative expression is easy and good.

Imperative is easily available as and when needed.

Can add dependency management prior to plugin use.

Should Gant have file relationship management as SCons, Waf, Rant, Make, etc. have?



Maven → Gant

This is a more complex issue.

What features should Gant provide to be seen as a viable replacement for Maven.

Need to have all the declarative nature.

Need to be able to use Gant to avoid writing specialist plugins.



Ganting a Maven Project

```
includeTargets << gant.targets.Maven
mavenCompileProperties = [ source : '1.5' , target : '1.5' , debug : 'true' , ]
mavenDependency << [ groupId : 'org.testng' , artifactId : 'testng' , version : '5.5' ,
    scope : 'test' , classifier : 'jdk15' ]

target ( test : "" ) {
    depends ( 'test-compile' )
    Ant.taskdef ( resource : 'testngtasks' ) {
        classpath {
            path ( refid : mavenDependencyClasspathId )
        }
    }
    Ant.testng ( outputdir : mavenTargetPath + '/test-reports' ) {
        classpath {
            pathelement ( location : mavenMainCompilePath )
            pathelement ( location : mavenTestCompilePath )
            path ( refid : mavenDependencyClasspathId )
        }
        classfileset ( dir : mavenTestCompilePath )
    }
}

target ( 'default' : "" ) { test ( ) }
```



Gant, Maven Style

```
group = 'org.codehaus.groovy.gant04.test'  
name = 'test1'  
version = 1.0  
lifecycle = new org.codehaus.groovy.gant04.JarLifecycle ()  
lifecycle.dependencies = [ './src/test/libs/commons-io-1.2.jar' ]  
lifecycle.testDependencies = [ './src/test/libs/junit-3.8.2.jar' ]
```

Hans Dockter
created this
prototype.



And the Result...

Clearly, using Groovy scripting beats writing in XML – no contest.

Ant's future is both secure and in doubt:

The Ant tasks are immensely useful – Gant wins.

Specifying in XML is a dead end – Gant wins.

The question is then how to replace Maven:

Plugin approach, or

Be a replacement.

If there is a threat to Gant, it is Buildr.

