



Maven, Eclipse and OSGi working together

Carlos Sanchez

March 17, 2008



About me

Member of Apache Maven PMC
Committer at the Eclipse Foundation
Director at Exist Global



Contents

1. Maven in the Eclipse IDE
2. Maven to OSGi
3. Eclipse plugins to Maven
4. Maven building Eclipse plugins
5. Looking into the future
6. Conclusions



Section 1

Maven in the Eclipse IDE



Maven in the IDE

Q4E

[Q for Eclipse]

Proposed to the Eclipse Foundation as
Eclipse IAM

[Eclipse Integration for Apache Maven]



Maven in the IDE

Other alternatives

m2eclipse

Maven Eclipse plugin

[eclipse:eclipse]



Q4E Features

running Maven goals from the IDE

dependency managing using the POM
& automatic download of dependencies

Eclipse classpath synchronized with POM



Q4E Features

direct import of Maven 2 projects

wizard for creation of new projects using
the archetype mechanism



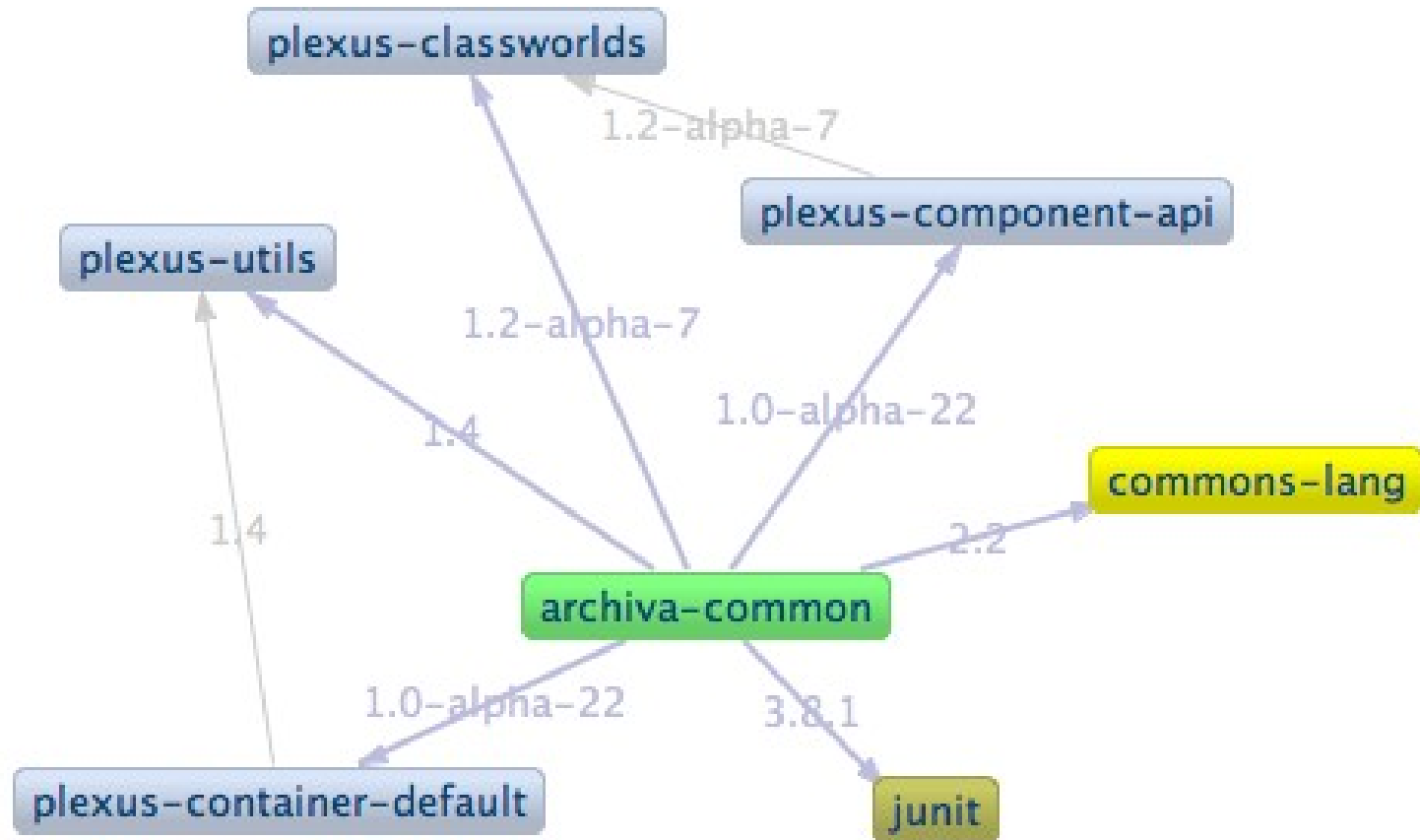
Q4E Features

modular approach to improve reusability
by other Eclipse projects

ability to import parent projects (pom
projects)

ability to cancel Maven builds

Q4E dependency graphing





Roadmap

WTP support

Integration with other plugins

Scala, Candy for Appfuse, SpringIDE,...

UI tooling to work around Maven
limitations (common complains)

Excluding a dependency from the graphs will add all
necessary <exclusions> to the pom



Q4E as example

All examples in next sections are
real-world examples
from Q4E development

<https://q4e.googlecode.com/svn/branches/mavenbuild>



Section 2

Maven to OSGi



Tools

Bleeding edge!!!

Apache Maven 2.0.9+

(includes bug fixes MNG-3396 and MNG-3410)

Apache Felix Bundle Plugin 1.4.1+

Maven PDE plugin 1.0+

Maven Eclipse plugin 2.5+

Converting 3rd party Maven projects to OSGi



Create a new Maven project

Set <packaging> to *bundle*

Add maven-bundle-plugin

Add dependencies to the 3rd party projects

Felix Maven Bundle plugin



Maven philosophy: will use defaults for most of the configuration

Manifest still completely customizable

<http://felix.apache.org/site/maven-bundle-plugin-bnd.html>

Felix Maven Bundle plugin



will scan your classes for
exported/imported packages

will not use Require-Bundle

Felix Maven Bundle plugin



By default `${groupId}.${artifactId}` packages get included, use `Export-Packages` to customize

`Export-Packages` exports AND includes the classes in the bundle

`<_exportcontents>` only exports

Felix Maven Bundle plugin



`inline=true` to avoid embedding jars
inside the bundle

Needs to run after compilation

Converting 3rd party Maven projects to OSGi



```
<packaging>bundle</packaging>
[...]
<build>
  <plugins>
    <plugin>
      <groupId>org.apache.felix</groupId>
      <artifactId>maven-bundle-plugin</artifactId>
      <!-- <version>1.4.0</version> defined in parent plugin management -->
      <extensions>true</extensions>
      <configuration>
        <instructions>
          <Embed-Dependency>*;scope=compile|runtime;inline=true</Embed-Dependency>
          <Embed-Transitive>>false</Embed-Transitive>
        </instructions>
      </configuration>
    </plugin>
  </plugins>
```

Converting 3rd party Maven projects to OSGi



Example

plugins/thirdparty/org.apache.maven.shared.dependency.tree

repackages

```
<dependency>  
  <groupId>org.apache.maven.shared</groupId>  
  <artifactId>maven-dependency-tree</artifactId>  
  <version>1.1</version>  
</dependency>
```

as OSGi bundle

Dependency handling



Customize the dependencies included

by using

<exclusions>

<_exportcontents>

<Export-Package>

<Embed-Dependency>

<Embed-Transitive>

Dependency handling



Customize the dependencies imported
and the versions

by using

Import-Package

Dependency handling:



example

plugins/maven/embedder/pom.xml

Includes all dependencies in a lib folder
inside the bundle

Some dependencies are exchanged for
prebuild OSGi bundles and depended
upon

Exports a few org.apache.maven and
org.codehaus.plexus packages only



Example

```
<dependency>
  <groupId>org.apache.maven</groupId>
  <artifactId>maven-embedder</artifactId>
  <version>2.1-SNAPSHOT</version>
  <exclusions>
    <!-- we'll use eclipse bundles for these -->
    <exclusion>
      <groupId>com.jcraft</groupId>
      <artifactId>jsch</artifactId>
    </exclusion>
    <exclusion>
      <groupId>aspectj</groupId>
      <artifactId>aspectjrt</artifactId>
    </exclusion>
  </exclusions>
</dependency>
[...]
```

```
<!-- Eclipse bundles dependencies -->
<dependency>
  <groupId>org.aspectj</groupId>
  <artifactId>runtime</artifactId>
</dependency>
<dependency>
  <groupId>com.jcraft</groupId>
  <artifactId>jsch</artifactId>
</dependency>
```

Example: Bundle plugin config

```
<plugin>
  <groupId>org.apache.felix</groupId>
  <artifactId>maven-bundle-plugin</artifactId>
  <configuration>
    <instructions>
      <_exportcontents>
        org.apache.maven.embedder.*;-noimport:=true,
        org.apache.maven.*;-noimport:=true,
        org.codehaus.plexus.util.xml.*;-noimport:=true,
        !org.codehaus.plexus.util.*,
        org.codehaus.plexus.*;-noimport:=true,
      </_exportcontents>
      <Export-Package></Export-Package>
      <Import-Package>
        !junit.*,
        !sun.misc.*,
        org.apache.commons.cli;version="[1.0.0,2.0.0)",
        *
      </Import-Package>
      <!-- we can't inline as the different plexus META-INF/plexus/components.xml
           will overwrite each other -->
      <Embed-Dependency>
        *;scope=compile|runtime;inline=false;artifactId=!cli|lang|runtime|tidy|jsch|
commons-logging|jdom
      </Embed-Dependency>
      <Embed-Directory>lib</Embed-Directory>
      <Embed-Transitive>true</Embed-Transitive>
      <Eclipse-BuddyPolicy>registered</Eclipse-BuddyPolicy>
      <Include-Resource>LICENSE.txt,NOTICE.txt</Include-Resource>
    </instructions>
  </configuration>
</plugin>
```

Converting your Maven projects to OSGi



Felix Bundle Plugin again

Generate the OSGi manifest

No need to change packaging

Same configuration options apply

Configure the jar plugin to pick up the generated manifest

Converting your Maven projects to OSGi



Using Apache Felix maven-bundle-plugin to generate the OSGi manifest

```
<plugin>
  <artifactId>maven-jar-plugin</artifactId>
  <configuration>
    <archive>
      <manifestFile>
        ${project.build.outputDirectory}/META-INF/MANIFEST.MF
      </manifestFile>
    </archive>
  </configuration>
</plugin>
<plugin>
  <groupId>org.apache.felix</groupId>
  <artifactId>maven-bundle-plugin</artifactId>
  <extensions>>true</extensions>
  <executions>
    <execution>
      <phase>process-classes</phase>
      <goals>
        <goal>manifest</goal>
      </goals>
    </execution>
  </executions>
</plugin>
```



Caveats

Version ranges have different meanings
in Maven and OSGi

Bundles with nested jars won't work

Private packages are visible

Version ranges in OSGi



$x.y.z.q > x.y.z$

org.eclipse.core.filebuffers

3.3.0 < 3.3.0-v20070606-0010

3.3.0-v20070606-0010 **is** in [3.3,4.0)

Version ranges in Maven



`x.y.z.q < x.y.z`

`x.y.z-q < x.y.z`

`org.eclipse.core.filebuffers`

`3.3.0-v20070606-0010 < 3.3.0`

`3.3.0-v20070606-0010` is **not** in `[3.3,4.0)`

Version ranges in Maven



History

1.0.0-SNAPSHOT = work in progress towards 1.0.0
1.0.0-SNAPSHOT becomes 1.0.0-20080301.101201-1
when deploying to the repository

1.0.0-SNAPSHOT < 1.0.0

then

1.0.0-20080301.1012 < 1.0.0

Version ranges in Maven



Solving the problem, from Maven 2.0.9!

use `<dependencyManagement>`
to explicitly set the versions you want
Do it in the parent POM and it's a one
time setup

Caveats: version ranges



```
<dependencyManagement>
  <dependencies>
    <!-- required for pde-maven-plugin due to problems with
         Maven and Eclipse version ranges -->
    <dependency>
      <groupId>org.eclipse</groupId>
      <artifactId>osgi</artifactId>
      <version>3.3.0-v20070530</version>
    </dependency>
    <dependency>
      <groupId>org.eclipse.ant</groupId>
      <artifactId>core</artifactId>
      <version>3.1.200-v20070522</version>
    </dependency>
    <dependency>
      <groupId>org.eclipse</groupId>
      <artifactId>text</artifactId>
      <version>3.3.0-v20070606-0010</version>
    </dependency>
    <dependency>
      <groupId>org.eclipse.core</groupId>
      <artifactId>commands</artifactId>
      <version>3.3.0-I20070605-0010</version>
    </dependency>
    <dependency>
      <groupId>org.eclipse.core</groupId>
      <artifactId>filebuffers</artifactId>
      <version>3.3.0-v20070606-0010</version>
    </dependency>
```

[...]

Caveats: Nested jars in bundles



The Sun compiler wont recognize them
Maven provides a pluggable compiler
infrastructure

Unfortunately the Eclipse compiler
implementation is not up to date

<http://maven.apache.org/plugins/maven-compiler-plugin/non-javac-compilers.html>

Caveats: private packages are visible



Same problem as with nested jars
The Sun compiler doesn't honor OSGi manifests



Section 3

Eclipse plugins to Maven

Converting Eclipse plugins to Maven



Maven Eclipse plugin 2.5+
eclipse:to-maven



Eclipse to Maven

Automatic process, bidirectional

Bundle-SymbolicName=x.y.z

groupId=x.y / artifactId=z

Bundle-Version=x.y.z.q

version=x.y.z-q



from OSGi manifest

```
Manifest-Version: 1.0
Bundle-ManifestVersion: 2
Created-By: 1.4.2 (IBM Corporation)
Bundle-Name: %PLUGIN_NAME
Bundle-Vendor: %PLUGIN_PROVIDER
Ant-Version: Apache Ant 1.7.0
Export-Package: org.eclipse.core.internal.net;x-internal:=true,org.ecl
ipse.core.net.proxy
Bundle-Version: 1.0.0.I20070531
Bundle-Activator: org.eclipse.core.internal.net.Activator
Require-Bundle: org.eclipse.core.runtime;bundle-version="[3.3.0,4.0.0)
"
Eclipse-LazyStart: true
Bundle-SymbolicName: org.eclipse.core.net;singleton:=true
Bundle-RequiredExecutionEnvironment: J2SE-1.4,CDC-1.0/Foundation-1.0,J
2SE-1.3
Bundle-Localization: plugin
```

plugin.properties

```
PLUGIN_NAME=Internet Connection Management
PLUGIN_PROVIDER=Eclipse.org
```





to Maven

POM

```
<?xml version="1.0" encoding="UTF-8"?>
<project>
  <modelVersion>4.0.0</modelVersion>
  <groupId>org.eclipse.core</groupId>
  <artifactId>net</artifactId>
  <name>Internet Connection Management</name>
  <version>1.0.0-I20070531</version>
  <licenses>
    <license>
      <name>Eclipse Public License - v 1.0</name>
      <url>http://www.eclipse.org/org/documents/epl-v10.html</url>
    </license>
  </licenses>
  <dependencies>
    <dependency>
      <groupId>org.eclipse.core</groupId>
      <artifactId>runtime</artifactId>
      <version>[3.3.0,4.0.0)</version>
    </dependency>
  </dependencies>
</project>
```



Eclipse to Maven: Caveats

Handling fragments
anything special to do?

Short names for artifactIds
not a problem in the repository
in a flat folder the Maven plugins should
automatically rename to
groupId.artifactId



Section 4

Maven building Eclipse plugins

Maven building Eclipse plugins



OSGi bundles

+

extra manifest values

(configurable in the Felix Bundle Plugin)

+

Features

+

Update sites



Integrating with PDE

Felix Bundle Plugin

generate the manifest in

```
${basedir}/META-INF
```

Jar plugin

use `${basedir}/META-INF` **as manifest**

source



Integrating with PDE

Dependency plugin

copy dependencies to local folder

binaries to `${basedir}/lib`

sources to `${basedir}/src`

Need to be added to Eclipse classpath by
hand

Integrating with PDE example



plugins/pom.xml

plugins/maven/embedder/pom.xml

Profile *eclipse-dev*

```
mvn -Peclipse-dev package
```



plugins/pom.xml

```
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-dependency-plugin</artifactId>
  <executions>
    <execution>
      <id>copy-dependencies</id>
      <phase>package</phase>
      <goals>
        <goal>copy-dependencies</goal>
      </goals>
      <configuration>
        <outputDirectory>${libdir}</outputDirectory>
      </configuration>
    </execution>
    <execution>
      <id>copy-src-dependencies</id>
      <phase>package</phase>
      <goals>
        <goal>copy-dependencies</goal>
      </goals>
      <configuration>
        <classifier>sources</classifier>
        <failOnMissingClassifierArtifact>>false</failOnMissingClassifierArtifact>
        <outputDirectory>${srcdir}</outputDirectory>
      </configuration>
    </execution>
  </executions>
</plugin>
```



plugins/pom.xml

```
<plugin>
  <groupId>org.apache.felix</groupId>
  <artifactId>maven-bundle-plugin</artifactId>
  <extensions>true</extensions>
  <configuration>
    <manifestLocation>${basedir}/META-INF</manifestLocation>
    <instructions>
      <Embed-Dependency>*;scope=compile|runtime;inline=true</Embed-Dependency>
      <Embed-Transitive>>false</Embed-Transitive>
    </instructions>
  </configuration>
  <executions>
    <execution>
      <phase>process-classes</phase>
      <goals>
        <goal>manifest</goal>
      </goals>
    </execution>
  </executions>
</plugin>
```

Maven building features



Add pde-maven-plugin, enable extensions

Set packaging to eclipse-feature

license must be present

Generates feature.xml and
feature.properties

Integrates with PDE by generating the
files in the root folder



Feature example

Add a dependency to previously
generated bundle

`org.apache.maven.embedder`

Exclude dependencies that are already
packaged in the bundle



Feature example

features/pom.xml

```
<plugin>
  <groupId>org.codehaus.mojo</groupId>
  <artifactId>pde-maven-plugin</artifactId>
  <extensions>true</extensions>
</plugin>
```

[...]

```
<profiles>
  <profile>
    <id>eclipse-dev</id>
    <build>
      <plugins>
        <plugin>
          <groupId>org.codehaus.mojo</groupId>
          <artifactId>pde-maven-plugin</artifactId>
          <configuration>
            <outputDirectory>${basedir}</outputDirectory>
          </configuration>
        </plugin>
      </plugins>
    </build>
  </profile>
</profiles>
```



Feature example

features/org.apache.maven.feature

```
<packaging>eclipse-feature</packaging>
<name>Apache Maven</name>
<description>Eclipse Feature for Apache Maven</description>

<licenses>
  <license>
    <name>The Apache Software License, Version 2.0</name>
    <url>http://www.apache.org/licenses/LICENSE-2.0.txt</url>
    <distribution>repo</distribution>
  </license>
</licenses>
<organization>
  <name>The Apache Software Foundation</name>
  <url>http://www.apache.org/</url>
</organization>
<url>http://www.apache.org/</url>

<dependencies>
  <dependency>
    <groupId>org.apache.maven</groupId>
    <artifactId>embedder</artifactId>
    <exclusions>
      <exclusion>
        <groupId>org.apache.maven</groupId>
        <artifactId>maven-embedder</artifactId>
      </exclusion>
    </exclusions>
  </dependency>
</dependencies>
```

Feature example: result



```
<feature
  id="org.apache.maven.feature"
  label="%featureName"
  version="2.1.0.632696"
  provider-name="%providerName">

  <description>
    %description
  </description>

  <copyright>
    %copyright
  </copyright>

  <license url="%licenseURL">
    %license
  </license>

  <plugin id="org.apache.maven.embedder" download-size="2262" install-size="2262"
    version="2.1.0.632696" unpack="false"/>

  <plugin id="org.apache.commons.cli" download-size="31" install-size="31"
    version="1.0.0.v200709131616" unpack="false"/>

  <plugin id="org.apache.commons.lang" download-size="206" install-size="206"
    version="2.1.0.v200709131643" unpack="false"/>

  <plugin id="org.aspectj.runtime" download-size="95" install-size="95"
    version="1.5.4.200705211336" unpack="false"/>
```

[...]

Feature example: feature.properties



```
#  
#Tue Mar 04 11:35:43 PST 2008  
featureName=Apache Maven  
copyright=Copyright DevZuz http://www.devzuz.org  
licenseURL=http://www.eclipse.org/org/documents/epl-v10.html  
license=Eclipse Public License - v 1.0  
description=Eclipse Feature for Apache Maven  
providerName=DevZuz
```



Maven building update sites

Add dependencies to the features you want to include

dependency type is eclipse-feature

To generate the site.xml and copy the features

Add pde-maven-plugin

Add the update-site goal to the execution

Maven building update sites



To copy all the bundles

Add Felix bundle plugin

Add the bundleall goal to the execution

it will process all dependencies,
dependencies' dependencies and so on

Maven building update sites



Set `ignoreMissingArtifacts` to true

From 1.4.1 you can limit the depth,
`depth=2` will do the trick

For integration with PDE, generate the
files in the root dir

Maven building update sites



Example

`updatesite-dev/pom.xml`

Has two features and its bundles

`org.apache.maven.feature`

`org.devzuz.q.feature.thirdparty`

Update Site example



```
<plugin>
  <groupId>org.codehaus.mojo</groupId>
  <artifactId>pde-maven-plugin</artifactId>
  <executions>
    <execution>
      <phase>process-classes</phase>
      <goals>
        <goal>update-site</goal>
      </goals>
    </execution>
  </executions>
</plugin>
<plugin>
  <groupId>org.apache.felix</groupId>
  <artifactId>maven-bundle-plugin</artifactId>
  <executions>
    <execution>
      <phase>process-classes</phase>
      <goals>
        <goal>bundleall</goal>
      </goals>
    </execution>
  </executions>
  <configuration>
    <depth>2</depth>
    <ignoreMissingArtifacts>true</ignoreMissingArtifacts>
    <outputDirectory>${project.build.directory}/site/plugins</outputDirectory>
  </configuration>
</plugin>
</plugins>
</build>
```

Update Site example



```
<dependencies>
  <dependency>
    <groupId>org.apache.maven</groupId>
    <artifactId>feature</artifactId>
    <version>2.1.0-632695</version>
    <type>eclipse-feature</type>
  </dependency>
  <dependency>
    <groupId>org.devzuz.q.feature</groupId>
    <artifactId>thirdparty</artifactId>
    <version>0.5.0.200803040914</version>
    <type>eclipse-feature</type>
  </dependency>
</dependencies>
```



Update Site caveats

Features must have a license, make sure you have one in your feature pom.xml

Eclipse update site manager will not give any detail if there are errors in features or update site



Section 5

Looking into the future

Equinox provisioning [p2]



Forget about features and update sites

Installable Units

new metadata format

external to the bundle

looks like a Maven POM? wait...

Artifact Repository



For OSGi bundles

sounds familiar?

It could be a Maven repository

When did you say?



Initial implementation in Eclipse 3.4

More to come in 4.0



Conclusions

Not ready for production

but

it will never be if people don't try it

Although it can already alleviate some
pain



Conclusions

Make your Maven projects be OSGi bundles at the source if you are going to need them later

Start generating manifests and committing them to work with both Maven and PDE



Conclusions

All the plugin versions used in the examples to be released soon



Homework

If you have free time
and want to contribute...



Eclipse compiler

You can configure Maven to use the Eclipse compiler and take full advantage of its OSGi support

<http://maven.apache.org/plugins/maven-compiler-plugin/non-javac-compilers.html>

The code

<http://svn.codehaus.org/plexus/plexus-components/trunk/plexus-compiler/plexus-compilers/plexus-compiler-eclipse/>

Issue tracker

<http://jira.codehaus.org/browse/PLXCOMP>

Thanks



carlos@apache.org

<http://www.carlossanchez.eu>