Technology management using artifact life cycles

Introduction and motivation

When I compare the Maven dependency mechanisms with our home-brewn solution in our company then among others one major thing is different: Maven does not know the concept of an artifact life cycle. (At least I do not know about such a mechanism and I do not refer to the build life cycle). Such life cycle status information would allow to extend the dependency management in a new dimension. One could declare whether certain dependencies are actually allowed to be used in a project, enabling effective technology management.

UPDATE/2008-06-19: The plugin has been made available on SourceForge: http://madp.sf.net

Use cases

Consider the following sample use cases.

Scenario 1: Flawed versions
It turns out that `my-app-1.4.2.jar` contains a serious security issue and is therefore flawed. Clients of this JAR should actually switch to a newer version `my-app-1.4.3.jar` which fixes the bug and which is safe to use.

Scenario 2: Decommissioning
Let's assume that `my-app-1.4.2.jar` is not supported anymore and projects should actually switch to a new release stream (`my-app-2.x.y.jar`).

Scenario 3: Restricted usage
Consider a library which has a restricted set of client projects, e.g. only certain projects are allowed to depend on a specific artifact.

On one hand, this life cycle information could be used to manage a repository in a more restrictive way, which makes it actually possible to perform technology management. On the other hand, when developers try to depend on an artifact which is actually not allowed, Maven could perform checks during the build life cycle and warn the user about inappropriate technology usage (dependency enforcement). Based on a flag the build would either fail or print a warning.

Our solution

Our solution works as follows. The technology board decides which versions of a dependency are actually allowed and this information is declared in an XML file:

```
<product name="struts">
  <version pattern="*" status="prohibited.not.investigated" />
  <version pattern="1.0*" status="prohibited.removed" />
  <version pattern="1.2.4" status="prohibited.flawed" comment="security issue
(bug:38374). upgrade to 1.2.9"/>
  <version pattern="1.2.4clx" status="approved.restricted" comment="technology preset
by APP1 release">
    <scope name="app1" />
  </version>
  <version pattern="1.2.9" status="approved.mainstream" comment="resolves security
(DOS) issue (bug:38374)" />
</product>
```

The build output would be as follows (solved with a simple Ant target):

```
init:
[echo] - status -- external project dependencies --------------
[depend] [ OK ] VALIDATOR_HOME='jakarta/commons-validator/1.1.3'
[depend] status 'approved.mainstream' defined for version pattern
'1.1.3': fixes dtd fetch issue of version 1.0.*, used in struts
[depend] [ OK ]  JAVA_HOME='/share/java/jdk/1.5.0_10'
[depend] status 'approved.mainstream' defined for version pattern
'1.5.0_10': Regard company guidelines for J2SE 5
[depend] [FAIL]  STRUTS_HOME='jakarta/struts/1.2.4'
[depend] status 'prohibited.flawed' defined for version pattern
'1.2.4': security issue (bug:38374). upgrade to 1.2.9
BUILD FAILED
Total time: 5 seconds
```

Solution in Maven

Would such an extension make sense in Maven? Software companies definitively have to solve their technology management and if they choose Maven for dependency management they could immediately benefit from such a feature. The question is if the open source community would benefit as well? I would say yes: just consider scenarios 1 and 2 above.

So how would this feature be implemented? I think that the appropriate file would be a file similar to `maven-metadata.xml`. I named it `maven-artifact-lifecycle.xml`. Here's a sample file for JUnit and please also note the XML schema attached.

```
<artifactLifecycle>
  <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <lifecycleStates>
      <lifecycleState>
        <versionPattern>3.8.1</versionPattern>
        <status>approved</status>
        <projectPatterns>
          <projectPattern>my.fancy.project.*</projectPattern>
      </projectPatterns>
    </lifecycleState>
  </lifecycleStates>
</artifactLifecycle>
```

I wrote a little Maven plugin called `maven-assertdepend-plugin` which I'm willing to contribute if there is interest. It checks against the information contained in the `maven-artifact-lifecycle.xml` of each dependency during the validation phase. Here's some sample output:

```
jogy@shadowfax:~/projects/assertdependtest> mvn install
[INFO] Scanning for projects...
[INFO]
------------------------------------------------------------------------
[INFO] Building Test project for maven-assertdepend-plugin
[INFO]    task-segment: [install]
[INFO]
------------------------------------------------------------------------
[INFO] [assertdepend:assert-depend {execution: assert-depend}]
[INFO] APPROVED: junit:junit:jar:3.8.1:test
[INFO]
------------------------------------------------------------------------
[ERROR] BUILD ERROR
[INFO]
------------------------------------------------------------------------
[INFO] DEPRECATED: com.company:util.jar:1.0:compile, comment: Use version
2.1 instead.

[INFO]
------------------------------------------------------------------------
[INFO] For more information, run Maven with the -e switch
[INFO]
------------------------------------------------------------------------
[INFO] Total time: 4 seconds
[INFO] Finished at: Wed Jan 02 22:07:41 CET 2008
[INFO] Final Memory: 4M/7M
[INFO]
------------------------------------------------------------------------
```

And here is some sample output using the warning switch:

```
jogy@shadowfax:~/projects/assertdependtest> mvn -Dassertdepend.warn=true
install
[INFO] Scanning for projects...
[INFO]
------------------------------------------------------------------------
[INFO] Building Test project for maven-assertdepend-plugin
[INFO]    task-segment: [install]
[INFO]
------------------------------------------------------------------------
[INFO] [assertdepend:assert-depend {execution: assert-depend}]
[INFO] APPROVED: junit:junit:jar:3.8.1:test
[WARNING] DEPRECATED: com.company:util.jar:1.0:compile, comment: Use
version 2.1 instead.
[INFO] [resources:resources]
[INFO] Using default encoding to copy filtered resources.
[INFO] [compiler:compile]
[INFO] Nothing to compile - all classes are up to date
[INFO] [resources:testResources]
[INFO] Using default encoding to copy filtered resources.
[INFO] [compiler:testCompile]
```

```
[INFO] Nothing to compile - all classes are up to date
[INFO] [surefire:test]
[INFO] Surefire report directory:
/Users/jogy/projects/assertdependtest/target/surefire-reports

-------------------------------------------------------
 T E S T S
-------------------------------------------------------
Running com.company.AppTest
Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.237 sec

Results :

Tests run: 1, Failures: 0, Errors: 0, Skipped: 0

[INFO] [jar:jar]
[INFO] Building jar:
/Users/jogy/projects/assertdependtest/target/assertdependtest-1.0-SNAPSHOT
.jar
[INFO] [install:install]
[INFO] Installing
/Users/jogy/projects/assertdependtest/target/assertdependtest-1.0-SNAPSHOT
.jar to
/Users/jogy/.m2/repository/ch/adnovum/assertdependtest/1.0-SNAPSHOT/assert
dependtest-1.0-SNAPSHOT.jar
[INFO]
------------------------------------------------------------------------
[INFO] BUILD SUCCESSFUL
[INFO]
------------------------------------------------------------------------
[INFO] Total time: 11 seconds
[INFO] Finished at: Wed Jan 02 22:08:06 CET 2008
[INFO] Final Memory: 5M/12M
```

```
[INFO]
------------------------------------------------------------------------
```

The Archiva subproject would probably be the best place to maintain this information because it supports user roles. Archiva could even check for *illegal* dependencies.