Packaging vs Type - Derived and Attached Artifacts

Summary of Solution

We need:

1. building an attachment (distribution primarily) must build the original, and deploy both.
2. type to packaging + classifier mappings (but no need for the reverse)
3. binding for the package step (eg ejb-client, tld) - this will also be applicable to deployment, snapshot or not
4. binding for the release step (eg sources, javadoc, distributions, etc)

Building an attachment must build both

As the assembly is blended into the lifecycle when a release is being performed, the packaging is already assured.

Type -> packaging + classifier mapping

Plugins can specify implementations of type handlers (and many will be in the core). These will indicate the packaging and classifier that they belong to. The default implementation will use packaging = type, no classifier.

Attachment at packaging step

Here, a simple call to project.attachArtifact(Artifact artifact) is appropriate. In the specific use cases:

- ear:ear would create the client based on configuration, and attach it
- a webapp plugin could have a goal, bound to packaging, that when configured published a set of tlds as well as bundled them into a jar

Attachment at deploy step

We want attachment of sources and javadoc to be the default, but we also want to configure in the attachment of various assemblies.

This will also depend on whether it is a snapshot build, or a release build as to what is actually attached. Profiles should help here.

We will introduce a a package-additional step after package, where this will occur (and they will use project.attachArtifact to trigger installation and deployment). Sources and javadoc will be enabled by default for jar:jar by the jar packaging lifecycle bindings. These will currently just check for the existence of SNAPSHOT as to whether they will do anything, but in future they should always do their task, and only be bound when in release mode (unless the user has specifically bound them somehow?)

Original Discussion - Differences Between Packaging and Dependency Type

Ref: http://jira.codehaus.org/browse/MNG-257.

I'd like to work through any problems that might arise from the differences between <packaging> and <type>. Below is the background, problem statement, and design decision to rectify them. Please add any additional problems you see, or issues with the design.

Background

In Maven 2.0, projects are keyed uniquely on only groupId and artifactId for considering what is a project (version does come into it for determining different versions of a project).
A packaging is specified on pom.xml to signify how to build that project, and what type of artifact it produces.

For a given packaging, it may produce multiple artifacts that share the same project information - the main example used is an ejb that also outputs an ejb-client. There are also some goals that will produce artifacts from a given project - for example, the assembly/distribution goals, javadoc and sources.

To be able to depend on only some of those artifacts, <type> on a dependency may not match packaging - if you declare <type>ejb</type> you only get the ejb, and if you declare <type>ejb-client</type>, you only get the ejb-client JAR. The POM can always be located by the groupId/artifactId/version combination (type is not present in the m2 repo, and the POM is always under /poms/ if using the old layout).

This also means that while type is not considered in whether an project is unique, it is for a dependency - so you can download tld files with the same name as the JAR they belong to.

This is pretty sound from general use cases, though there have been a couple of problems to arise:

- inconsistency could lead to confusion
- guaranteeing snapshots match versions

Inconsistency:

I believe this is ok. The naming was different to ensure it is clear that they can be different, and packaging IS A SUBSET of <type>, guaranteeing they can be handled consistently. Any value of packaging should have 1 type that matches exactly, and possibly more types that are associated with it. Any type has 0..1 matching packaging.

Snapshots:

If a JAR is deployed, then a distribution is deployed - either the POM is republished (so the snapshot is bumped up and the JAR is orphaned), or it is not and the published dist is orphaned, or reuses the timestamp which is not technically correct (especially as it may overwrite a previous distribution).

Discussion

Firstly, we differentiate types in the following way:

- that specified by packaging
- other types "tied" to the packaging (eg ejb-client and ejb)
- other types only generated by particular optional goals (eg distribution)

Untied objects can be manually attached, and tied objects can be manually detached. From this, it seems sensible that attachment is only a matter of configuration, with sensible defaults, rather than something specific in the design. Note that there are different scenarios: attachment by configuration is appropriate for ejb client, but when set it should always happen. For distributions, sources, etc it is appropriate to only attach at release - they do not need to be deployed for snapshots.

- deploying a distribution MUST deploy the JAR - ie, any deployment must deploy both the POM and the artifact of the packaging (+ anything tied to it, like an ejb-client or sources).
- deploying an artifact MUST deploy any "tied" artifacts - eg, deploying an EJB must deploy its client
- deploying an artifact NEED NOT deploy any "untied" artifacts - eg, deploying a JAR doesn't deploy the distribution by default - it must be configured to do so, or the goal added

This ensures that a POM is always deployed and updated for every deployment of anything, and that the primary artifact and anything tied to it is never orphaned.

It can mean that under the default lifecycle, there will be no snapshots deployed for a distribution, for example. This simply means you cannot depend on a distribution where you can a JAR, unless it is tied, which is reasonable.

This does not affect released artifacts, as they can be released at different times and still have the same version.

Declaring the type in a dependency

It is envisaged that the type could encapsulate the extra information about the classifier. ie, ejb-client is a type that maps to the packaging ejb with the classifier "client".

However, there are a few interesting points here:

- have decided to make groupId/artifactId unique, regardless of type. Therefore, type is not required to lookup an artifact at all (though it would be helpful in indicating what type the artifact is when reading the pom)
- the above could mean that type could be replaced by by something solely to identify the classifier
- if not, then either the type needs to map to a packaging + classifier, or a classifier field needs to be added
- if the type is being mapped, this needs to be quite flexible - ie a plugin can provide its own, but this means that arbitrary classifiers can't be used. This has positives and negatives of its own.

I am in favour of using type mappings, at least to start with. The positives are:

- avoids arbitrary classifiers, making artifactId be well considered
- avoids requirement of additional metadata field in the dependency element

Drawbacks:

- can't create your own classifiers, must stick to those provided by the plugin producing the artifact
- must build in the mapping to all plugins providing a packaging.