Specification Dependencies and Provides Notation

Implementing Specification Dependencies and Provides Notation

Context

It's possible for a given API to have multiple implementations. For example, the JavaMail API has the reference implementation provided by Sun, along with multiple open source versions. In many cases, POM authors will want to specify that a project depends on JavaMail, but won't necessarily want to dictate which implementation of the API is used. Moving to specification dependencies will solve this problem. The specification POM will serve as a pointer for a set of libraries which implement the API. The specification dependency would then be resolved in a number of ways, possibly with user preferences factored in via settings.xml.

Provides notation would be related to specification dependencies, but on the opposite side of the artifact relationship. For instance, Sun's JavaMail implementation would specify that it provides an implementation of the JavaMail API, as would the javamail library built by Geronimo. It would act as a reverse mapping of which API(s) are provided by a given artifact. This provides specification might need to be plural, to allow a single artifact to implement multiple APIs, thereby satisfying multiple specification dependencies from a single build.

In addition, we need a mechanism for providing a provides-ish notation for assemblies and other attached artifacts that may contain one or more of the project's dependencies. For example, if a project's build produces an assembly using the 'jar-with-dependencies' pre-defined assembly descriptor, it should have some way of indicating to Maven that it already contains those dependencies. This way, it can be used as a substitute for other dependencies specified by a user, if those dependencies have the same groupId:artifactId:version:classifier:type. **If there is a mismatch of dependency IDs between those provided by the assembly and those specified directly in the user's POM, then the assembly should not be allowed in the build.**

Problem

Maven currently has a very concrete notion of what constitutes an artifact. In many ways, an artifact is assumed to be a physical, binary file out on one or more repositories. We have flirted with some forms of virtualization by supporting snapshot dependencies, RELEASE/LATEST meta-versions for plugins, and POM relocations. However, we currently have no way to handle 1:N mappings of a specified dependency to potentially satisfying artifacts.

Specifically, Maven lacks two things to support this concept:

1. Maven's artifact-resolution code must be able to determine implementation-preference when multiple candidates exist
2. POMs need some way to articulate which abstract POM they "implement" (i.e. which specification(s) they satisfy)

Once we have these two pieces, the specification-POM becomes simply another layer of metadata, much like a RELEASE version for a plugin.

Resources