This proposal describes a fourth rewrite of Maven's artifact and artifact-transfer APIs.

Primary Motivation

With Maven 3.0, the artifact APIs have become a thin wrapper around the Eclipse Aether project APIs and implementations. So thin, in fact, that Aether imports are sprinkled throughout Maven itself, and they are still spreading (now, through plugin code). This represents a coup of sorts for Aether, in that it removes some of Maven's ability to continue innovating. For example, implementing a group- and mirror-based routing implementation of the artifact APIs for Maven has become extremely difficult, not least because it requires convincing the Aether project's management of its value. At the very least, it's important to get Maven back to a point where Aether is an implementation detail, and not part of the imports used throughout plugins and Maven itself. This will allow Maven to continue evolving without requiring approval from external projects.

Beyond this, Maven has become widely adopted, to the point that the Maven concepts of Artifacts and Repositories now provide the de facto standard for stitching together projects with their dependencies during java builds. Yet many of the advanced potential for working with artifacts and repositories are limited by the APIs available to process them. Prominent here is the difficulty in building a full dependency graph containing dependencies, parent metadata (POMs), plugins, build extensions, etc. Maven already builds up dependency graphs for all of these things when it constructs the build runtime, but it doesn't provide a good way for interacting with the graph once built. Providing depgraph capabilities will improve the potential for error reporting within Maven itself, as well.

Additionally, doing a full rewrite allows us to re-examine some of the concepts codified in the API, to learn from and improve some of the things that have caused confusion in past Maven versions.

This rewrite should address the following points:

Goals

No Thirdparty Imports

We need to avoid exposing ANY thirdparty APIs via imports or components to be used by plugins, extensions, etc. in order to maintain (restore?) control over our own legacy.

Revise Transfer Verbs

'install' as a verb in a build process is misleading. We should consider using 'cache'.

'deploy' is also not a build activity, and also misleading; consider 'publish' or similar as an alternative.

Local Repository Clean-Up

The local repository seems too much like a remote repository. We should consider referring to it as a 'cache', and making it less accessible for abuse. Additionally, we need more control over what content lands in the cache, when, and how it's organized.

We should consider the following:

- allow Maven to skip caching altogether (meaning a build-session cache only)
- separate content by repository source
- provide a means of cleaning out cached artifacts by source, etc.

API / Interface Design

We need to learn from the limitations of past incarnations of the artifact/repository API. Major functions should be contained in their own 'collecting' interface (this is better in Aether). Also, API classes should guide the user to supply the minimal information required for the class to be functional. For example, a resolution request should not allow the list of artifacts to be empty.
Implementations should be easy to replace with new providers. We've always had in mind the notion of adding new remote repository implementations, and even our MirrorSelector is an interface, implying a desire to allow alternative implementations. We need to spend time thinking of how to allow users easy access to substitute in these alternatives, and give them a good bottleneck interface they can replace. A great example is in the router-based resolution I've tried to implement. I have an API for looking up the repository/mirror URL per groupid, allowing each project (usually publishing its own groupid) to manage a canonical repository URL and set of mirrors that contain that groupid. Maven should be able to use this information to select which repositories to request artifacts from, and eliminate the need for a centralized Maven repository altogether. However, plugging this routing API into Maven 3 is a nightmare, simply because of all the places where default components must be replaced (still not a particularly easy task in Maven 3).

**Expose (Immutable) Graph Information**

All dependency graph information is static; therefore, the depgraph/artifact implementations should be immutable as well. Scope shift in N+1 dependency layers and such should be handled through a flexible scope-selecting filter or graph traverser. These graph structures should be exposed via the API for plugins and third parties to use in analyzing project interconnections.

**Error Handling**

We need to design for responses that return multiple errors across the board for anything that processes a group of information at once.

But the larger issue here is how to implement consistent error-handling design in the context of multi-error methods.