Interactive Dependency Graphs

Introduction

As a Google Summer of Code participant, this summer I'll be working on a dependency graphing plugin, with Carlos Sanchez as mentor. See http ://docs.codehaus.org/display/MAVEN/Dependency+Graphing for background.

This page describes the necessary design to support interactive dependency graphs. This is on a separate page from the main Maven Diagram Maker page for now because it remains to be determined whether the UI should be merged, or if there is even a common core of classes to use.

Use Cases

The maven-grafo-plugin will allow Maven users to visually explore a project's dependency structure and gain the necessary information to fix dependency problems and effectively refactor the hierarchy of dependencies.

Brett Porter identified the following uses:

- Identifying dependency exclusions. Is it excluded, and where did exclusion originate.
- Identifying dependency version decisions. If maven picks another version, which one, and why?
- Identifying dependency owner. If dependency is present in project, how did it get there? By current project, by transitive dependency, or by parent pom?
- Identifying transitive dependency complexity. Multi module project is getting out of hand, what dependencies can i collapse safely into one dep?
- Identify relocated dependencies.
- Identify missing or bad dependencies.

Milos Kleint also identified:

- optional persistence of the visual data
- filters
- finding specific artifacts in the tree
- getting more information about artifacts
- answering "why was this artifact included?
- answering "why it changed scope or version?".

To help manage larger graphs, the following UI tasks will also be possible:

- Collapse a subtree: hide dependencies
- Change layout (force-directed, fixed, etc.)
- Pin an artifact to a particular location

User Interface

The UI is based around two goals in the grafo plugin, one to generate a file, one to display the UI:

grafo:generate

Generate a file containing a dependency graph. Parameters:

Parameter	Sample Value	Description
format	graphml	File format, corresponds to a sink.that generates a graph in String format.
scope	foo	Name of the scope, defaults to "compile".
file	foo.graphml	Name of the file to save. Required.

grafoApplet

If time permits, a Java applet will be created. This will be a simplified version of the grafo:ui except that the graph source will be an XML file, and parameters such as dependency scope and filters will be specified via applet parameters. The applet will not incorporate a menu bar.

grafo:ui

Display the current project dependencies in a GUI window.

GUI: Main Window

The UI for the Java application will consist of a menu bar and a Prefuse frame. The menus are as follows:

File menu

- **...Open** Open an XML graph file
- **...Save** Save the current graph in XML or other format

Artifact menu:

- **...Pin** Pin (unpin) a node to particular X/Y coordinates
- **...Collapse** Collapse (expand) the selected artifact's dependencies to simplify the graph
- **...Properties** Show a properties window

Graph menu

- **...Scope** Show the graph for a particular scope (compile, test, etc.)
- **...Filter** Filter artifacts for groupId/artifactId/version containing specified text
- **...Show Optional** Show/hide optional dependencies
- **...Show Excluded** Highlight/show/hide excluded dependencies(default: hide)
- **...Version Conflicts** Highlight all artifacts with more than oneversion in the graph (selected by default)
- **...Highlight Relocated** Highlight all relocated dependencies.
- **...Highlight Missing** Highlight missing/bad dependencies (selected bydefault)
- **...Compare** Compare the current view to a GraphML file. The GraphML file will be treated as the "old" version.
- **...Switch Layout** Switch to a different Prefuse layout. The available layouts are implementations of prefuse.action.layout.Layout.

Help menu:

- **...About** Show the grafo-plugin version

The user may also right-click on an artifact in the graph to display the Artifact menu.

GUI: Properties window

The properties window for an artifact identifies the following in a dialog box with a Close button:

- Group, project, version, artifact type
- Repository URL
- Scope
- List of all artifacts that depend on this one (vertical scroll if necessary)
- List of all artifacts that this depends on (vertical scroll if necessary)
- Whether the artifact is optional
- Included or skipped, and if skipped, why (in English). For example, "This artifact is omitted in favor of because plexus-utils 1.4 (a dependency of foo:bar 1.0) was nearer.
- Relocation details. One of "This artifact was not relocated", "This artifact was relocated to group:artifact:version", or the relocation message in the POM.

Class Design

A dependency graph is essentially a directed graph of org.apache.maven.artifact.Artifact objects. A source/sink model is used so that any representation of the graph can be translated to any other representation. What follows are the model classes; the UI classes are low complexity and will be "as necessary" to implement the UI specified above.

The normal way to visualize the data will be to connect the Maven source to the Prefuse sink. grafo:generate will connect the GraphML source to the user-specified sink. Comparison will involve connecting Prefuse and GraphML sources to a Prefuse sink (to merge the data).

IArtifactGraphSource

The Artifact Graph Source presents artifacts to sinks.

```
registerSink(IArtifactGraphSink sink)
unregisterSink(IArtifactGraphSink sink)
```

IArtifactGraphSink

This is an interface for dependency sinks, which are used to generate a
directed graph in a particular format for presentation.

```
clear()
```

Remove all nodes from the graph.

```
add(org.apache.maven.artifact.Artifact parent, org.apache.maven.artifact.Artifact
child)
```

Add an edge from parent to child. The child node is created if itdoes not exist.

```
setDetaultProperty(String name, Object value)
```

Set a property on all Artifact nodes that are referenced in an add action.

```
unsetDefaultProperty(String name)
```

Stop setting a property on referenced nodes.

```
setProperty(Artifact artifact, String name, Object value)
```

Specify additional information about an artifact that was determined by the sink, for example whether the dependency is skipped for a version conflict, locked position on the graph, etc. Not all sources will set all properties.

```
setGraphProperty(String name, Object value)
```

Specify a property that applies to the graph as a whole

```
Object getGraph(String scope)
```

Return the graph for the specified scope. If scope is null, the entire graph is returned. The specific representation of the graph is particular to the sink.

```
void saveGraph(File outputFile)
```

Serialize the graph to the specified output file.

TBD: How do we know when processing is started/complete?

GraphSinkRegistry

The graph sink registry maintains references to the sinks that can write meaningful file formats.

```
static void register(IArtifactGraphSink sink, String formatname, FileFilter
fileFilter)
```

Register a sink. formatname is the short format name ("png"), and fileFilter is the corresponding FileFilter for use with a JFileChooser.

MavenArtifactGraphSource

This source presents artifact dependencies identified during Maven's artifact resolution process.

GraphMLArtifactGraphSource

This DependencySource is used to read in a GraphML file.

GraphMLArtifactGraphSink

This is used to create a GraphML file.

PrefuseArtifactGraphSource

This source creates Artifact objects from a prefuse.data.Graph.

PrefuseArtifactGraphSink

This sink is used to create a prefuse.data.Graph of objects.