Felix Goes to Tuscany

Applying OSGi modularity after the fact

Luciano Resende
lresende@apache.org

Graham Charters
charters@uk.ibm.com
About Apache Tuscany

• Tuscany provides a component based programming model which simplifies development, assembly and deployment and management of composite applications in SOA.

• Apache Tuscany implements SCA standards defined by the OASIS OpenCSA and also provides extensions based on real user feedback.
About Apache Felix

- Apache licensed open source implementation of OSGi R4
  - Framework (in progress, stable and functional)
    - Version 1.2.1 currently available
- Implements additional services
  - OSGi Bundle Repository (OBR)
  - IPOJO - POJO-based component model
  - Maven Bundle Plugin
  - ...
Tuscany Environment before OSGi

- Modularization inspired in OSGi
  - 150+ Modules

- Multiple Extensions with different levels of dependencies
  - 120+ 3rd Party Dependencies

- Maven based build
Motivation for OSGi

• Better class loading mechanism for our modules
• Create clean boundaries between sub-systems
• Facilitate embedding Tuscany in OSGi based environment
• Without OSGi Java modularity is broken
  – OO modularity too fine-grained
  – Severely limited package modularity
  – Jars have no modularity characteristics
  – Classpath ordering defines which class you get
OSGi & SCA

• Support OSGi as a packaging mechanism for SCA application artifacts (contributions)
  – SCA specification already mentions OSGi as package skin
  – Leverage OSGi import/export to import java artifacts from different SCA application artifacts (contributions)

• Support OSGi as an SCA Component Implementation Type
  – Use SCA to assemble OSGi Bundles with other implementation technologies
Constraints

- No free-reign to drive through the changes
- Community Concerns:
  - Must not cease non-OSGi support
  - Must not significantly increase distribution footprint
  - Must not significantly increase build time
  - Must not significantly increase runtime costs
  - Must not overburden non-OSGi community
- These constraints influence speed of and approach to OSGi adoption
Supporting Tools

• We have found various tools available
  – Dependency analyze tools
    • BND
  – Bundle dependency visualization
    • <coderthoughts /> - GMF
    • <coderthoughts /> - ManyEyes
  – Maven related tools
    • Various maven plugins

• Our experience
  – In general, most of the tools have particular issues that didn’t allow us to have a fully OSGi experience
Dependency Analysis Tools

• BND
  – Tool for creating Bundles
  – Analyzes code to determine dependencies
  – Supports directives to tailor OSGi Manifest
  – Supports many build options
    • Command Line
    • Ant
    • Maven
    • Eclipse

http://www.aqute.biz/Code/Bnd
Apache Felix Maven Bundle Plugin

• The 'glue' between Maven and BND

```
...<plugin>
  <groupId>org.apache.felix</groupId>
  <artifactId>maven-bundle-plugin</artifactId>
  <configuration>
    <instructions>
      <!-- Bundle versioned from Tuscany version -->
      <Bundle-Version>${tuscany.version}</Bundle-Version>
      <!-- Bundle Symbolic name -->
      <Bundle-SymbolicName>org.apache.tuscany.sca.api</Bundle-SymbolicName>
      <!-- Bundle description from pom description -->
      <Bundle-Description>${pom.description}</Bundle-Description>
      <!-- Export org.osoa.sca and all sub-packages -->
      <Export-Package>org.osoa.sca*</Export-Package>
      <!-- No Import-Package so calculate imports from code dependencies -->
    </instructions>
  </configuration>
</plugin>...
```

Apache Felix Maven Bundle Plugin - Caveats

• Test dependencies are ignored during calculation of imported packages
  – Issues when tests have references to external packages

• Current solution
  – Created maven plugin that consider test dependencies and properly find import packages and mark them as optional
Bundle dependency visualization

- `<coderthoughts /> + GMF`
  - ASL2 licensed output from a blog by `<coderthoughts />`
  - Uses EMF to model and save Bundle runtime dependency resolution
  - Introspector bundle analyzes and saves dependencies from a running system
  - Uses GMF for Visualization

<coderthoughts /> + GMF

- Dependency analysis works very well
- GMF visualization does not scale!
ManyEyes

- IBM AlphaWorks shared data visualization service
- Visualization options include
  - Maps, Line Charts, Pie Charts, Tree Maps, Network Diagrams, and many more
- Used Network Diagram to visualize dependencies
- DataSet is simple table of dependant to dependee
  - Can use 'cat', 'grep' and 'sed' to slice-n-dice the data and experiment with combining Bundles
- <coderthoughts /> dependency analysis used to create DataSet

<table>
<thead>
<tr>
<th></th>
<th>Dependent</th>
<th>Depends on</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>contribution.xml</td>
<td>assembly</td>
</tr>
<tr>
<td>2</td>
<td>contribution.xml</td>
<td>contribution</td>
</tr>
<tr>
<td>3</td>
<td>contribution.xml</td>
<td>monitor</td>
</tr>
<tr>
<td>4</td>
<td>xsd</td>
<td>assembly</td>
</tr>
<tr>
<td>5</td>
<td>host.jetty</td>
<td>extensibility</td>
</tr>
<tr>
<td>6</td>
<td>host.jetty</td>
<td>host.http</td>
</tr>
<tr>
<td>7</td>
<td>host.jetty</td>
<td>core.spi</td>
</tr>
<tr>
<td>8</td>
<td>data_binding fasta</td>
<td>data_binding</td>
</tr>
<tr>
<td>9</td>
<td>data_binding.sdo</td>
<td>assembly</td>
</tr>
<tr>
<td>10</td>
<td>data_binding.sdo</td>
<td>contribution</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

http://services.alphaworks.ibm.com/manyeyes/home
Visualizations: Apache Tuscany OSGi Runtime dependencies (no 3rd party and no versions)

Can't see the visualization? Download the latest Java plugin here. On Macs: best viewed in Safari.

Created by: Graham Charters  Created on: Thursday September 04, 6:54 AM

Pan: right-click and drag
Zoom: left-click and drag or use mousewheel

http://services.alphaworks.ibm.com/manyeyes/view/15h8QsOtha6MtkkFzD9Q2~

Leading the Wave of Open Source
Maven builds and OSGi

- Maven 2.0.9+
  - Fixes for MNG-3396 and MNG-3410
  - Fixes that allow definition of specific dependency version when dependency range was defined.
Maven builds and OSGi - Caveats

• Version ranges have different meanings in Maven and OSGi
  – OSGi
    • x.y.z.q > x.y.z
    • 3.3.0 < 3.3.0-v20070606-0010
    • 3.3.0-v20070606-0010 is in [3.3,4.0)
  – Maven
    • x.y.z.q < x.y.z
    • x.y.z-q < x.y.z
    • 3.3.0-v20070606-0010 < 3.3.0
    • 3.3.0-v20070606-0010 is not in [3.3,4.0)

• 1.0.0-SNAPSHOT = work in progress towards 1.0.0

• Workaround
  – use <dependencyManagement> to explicitly define the version to be used
  – Requires maven 2.0.9+
Maven Eclipse Plugin

• Used to generate Eclipse IDE Files for given maven projects
  – *.classpath
  – *.wtpmodules
  – .settings folder
  – etc
Maven Eclipse Plugin - Caveats

- Eclipse plugin add dependency jars directly in the project classpath in addition to the “eclipse bundle class path container”

- Current solution
  - Created maven plugin to properly configure project classpath to use the “eclipse bundle class path container” and avoid adding the dependency jars directly to the classpath
Maven eclipse compiler

- The Sun compiler is not aware of OSGi Import/Export

- The maven-eclipse-compiler plugin allows us to directly use the Eclipse compiler that have better support for OSGi bundles
Maven eclipse compiler - Caveats

- We found various issues with the eclipse compiler plugin
  - Warnings would cause plugin to hang

- In progress solution
  - Using a forked version of the maven-eclipse-compiler plugin
  - Bring-up plugin to working stage
  - Enhancing to enforce OSGi Import/Export
Applying OSGi to Tuscany
One Big Bundle of Joy

- Recommended practice when moving to OSGi*
  - Create one big bundle containing application and dependent libraries
  - Get it working in OSGi
  - Gradually replace dependent libraries with Bundles
  - Keep it working!

- This is how we started...
  - 1 Bundle ~ 60MB made from 200+ jars

Decomposition First Attempt

- Identified five categories of jars and created corresponding Bundles
  - `org.apache.tuscany.sca.api.jar` 18,701
  - `org.apache.tuscany.spi.jar` 430,563
  - `org.apache.tuscany.runtime.jar` 538,660
  - `org.apache.tuscany.extensions.jar` 1,374,045
  - `org.apache.tuscany.depends.jar` 57,872,558

- Issues:
  - Too coarse-grained to be of real value
  - No opportunity for sub-setting
  - Not modular
Re-using Existing Decomposition

- Tuscany already decomposed into many Maven modules
- Benefits:
  - Maven Bundle Plugin makes it easy to create Bundles
  - Matches community's existing understanding
  - Same bundles can be used outside OSGi
  - Easily sub-set as Tuscany intended
- Issues:
  - Lots of classloader issues
    - Assumed single classloader
  - Difficult to consume (200+ bundles)
Granularity

- 200+ bundles cumbersome
- Multiple bundles required to enable one capability
- Much debate about right level of granularity
- Conclusion
  - Fine-grained bundles suitable for developer view
  - Features used to aggregate bundles to provide a user view
    - Inspired by Eclipse Features
Third-party Libraries

- Many third-party libraries not enabled for OSGi
- Repositories are emerging
  - OSGi Bundle Repository (OBR)
  - Apache Felix Commons
  - Eclipse Orbit
  - SpringSource Bundle Repository
- Tuscany has ~120 pre-requisite third-party libraries
- Version and footprint constraints influence choice of approach
  - Project not comfortable to go with repository choice
Third-party Library: wrap

- **Wrap the Jar in a Bundle**
  - Bundle-Classpath: third-party.jar

- **Pros**
  - Works for signed Jars
  - Can aggregate multiple Jars

- **Cons**
  - Jar no longer works in non-OSGi environment (doubles the build footprint)
Third-party Library: convert

- Convert the Jar to a Bundle

Pros
- Jar works in non-OSGi environment (no footprint issue)

Cons
- Doesn't work for signed Jars
- May affect library licensing
- Can't aggregate multiple Jars
Third-party Library: virtual bundle

- Convert Jar to a Bundle at runtime
  - Manifest pre-generated or created on-the-fly

- Pros
  - Jars completely unchanged
  - Works for signed Jars

- Cons
  - No 'real' bundle to work with during development
  - Messy – two artefacts to manage
Third-party Library: Unpacked wrap

- Unpacked wrap style bundle
  - Bundle-Classpath: third-party.jar

- Pros
  - Works for signed Jars
  - Can aggregate multiple Jars

- Cons
  - Dynamic resolving might have performance implications
    - Working on enhancing the tools to use BND logic to calculate import packages

MANIFEST
Bundle-Classpath: third-party.jar
Dynamic-Import-Package: *
Export-Package: ...
...

Install
OSGi Framework
OSGi Versioning

- Package exports can specify a version
- Package imports can specify a version range
- The OSGi resolver 'wires' imports to exports
**Versioning**

**The Idealist**
- Version range [1.5.0, 2.0.0)
- Flexible
- Relies on others to do the right thing
- Risky
- Makes an untested support statement

**The Realist (paranoid)**
- Fixed version [1.5.0, 1.5.0]
- Inflexible
- Will get the version you tested against
- Safe
- Inhibits bundle updates

**Tuscany community chose to start with fixed versions with a view to introducing ranges through experience**

“Apache Commons has guidelines, we should trust them to do the right thing.”

“Without the testing, we can’t be sure of anything.”
Extension Registry Pattern

- Module declares extension point
- Modules contribute extensions which implementation extension points
- Extension Registry manages extension point and extension matching
- Used extensively in Eclipse (not standard OSGi and not part of Felix)

http://www.eclipsezone.com/articles/extensions-vs-services/
Tuscany Extensibility

- OSGi optional so Tuscany needed its own thing
  - inspired by Extension Registry
- Tuscany SPI defines extension points
- Extension Modules contribute
  - Bindings (REST, json-rpc, SOAP, ...)
  - Implementation Types (POJO, BPEL, OSGi, ...)
  - Interface Types (Java, WSDL)
Summary

• It is indeed possible!
  – OSGi effort is making good progress

• Current Approach
  – Tuscany Modules \rightarrow OSGi Modules
  – 3rd Party Libraries \rightarrow OSGi Modules
    • Using Unpacked wrap style bundle
  – Bundle Manifests available in source repository and tweaked for optional test dependencies
  – Tools are still an issue
    • Have already created several toolings
    • Looking for a maven-eclipse-compiler that would enforce OSGi import/export
Useful Links

- Apache Tuscany
  - http://tuscany.apache.org
- Apache Felix
  - http://felix.apache.org
- Eclipse Equinox
  - http://www.eclipse.org/equinox/
- OSGi Alliance
  - http://www.osgi.org
- OSGi Best Practices
- Converting (Large) Applications to OSGi