Adding Modularity Afterward with Embedded OSGi
About Me – Bob Paulin

- @bobpaulin/bob@bobpaulin.com/http://bobpaulin.com
- Independent Consultant
  - Web Centric Platforms
  - Business Enablement
  - Continuous Delivery
- Chicago Java Users Group (CJUG) Community Leader
  - Need a Mentor? mentors@cjug.org
  - Want to Present in Chicago? present@cjug.org
- Proud Father/Husband with 3 kids (and a Cat since Developers all seemed to like cats)
Regret
Application Development Cycle

I can do anything!!!

Wow that was easy let's add more features

Design Pivot

Start Coding

Great Idea

Hmm how did that break

Wait we can't change that

If you change that we have to test the whole app

Please don't touch anything without running it by me first
Hope?
If we could start over what would we want?
Modularity
Options?
OSGi

- Mature 10+ Years
- Tools
- Modularity is enforced
- Versioning
- Complex Classloading
- Runtime
Inversion of Control (IoC)

- Lightweight
- Easily added to existing systems
- Tools
- Modularity not enforced (DIY)
- No Versioning
Jigsaw

- Java 7...8...9??
- Versioning
- Interoperability with OSGi (Penrose)
OSGi: Start with One big Bundle?
OSGi: When does embedding make sense?
Want an Isolated Third-Party Container
Working with OSGi unfriendly Libraries.
Licensing/Proprietary Code
No

Budget/Time/Desire

for full OSGi
Embedded Design/Implementation

Diagram:

- Embedded OSGi Runtime
- Bundle
- Initialize
- Provides Services to

Table:

<table>
<thead>
<tr>
<th>Existing Application</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Service Interfaces</td>
<td>org.mycompany.services</td>
</tr>
<tr>
<td>Framework Initialization</td>
<td></td>
</tr>
<tr>
<td>Service Accessor</td>
<td></td>
</tr>
<tr>
<td>Bundle Directory</td>
<td></td>
</tr>
</tbody>
</table>
Exposing packages to the framework via System Packages
Minimum required bundles

- Felix Framework
A few more to consider...

- Felix Config Admin
- Gogo Shell
- Web Console + HTTP
- SCR + Annotations
- Apache ACE Management Agent
Good places to start

- Configuration
- Factories
- Resources
Caveats

- Framework creates threads
- Use only one Framework Instance
- Requires some IDE tricks
- Package Tangling may get worse before it gets better (Use Sonar)
Talk is Cheap.

Time for an example.
An Embedded Usecase With Tanks

- **SOAR 2D Grid Game**
- **Written in C++ with Java Bindings**
- **Code is Coupled**
- **SWT**
- **Multiple Games**
Exposing packages to the framework via System Packages

```java
configMap.put(Constants.FRAMEWORK_SYSTEMPACKAGES_EXTRA,
    "edu.umich.soar,edu.umich.soar.gridmap2d.config," +
    "edu.umich.soar.gridmap2d,edu.umich.soar.gridmap2d.world," +
    "edu.umich.soar.gridmap2d.visuals," +
    "org.eclipse.swt.widgets,org.eclipse.swt.graphics; version=0.0.1");
```
Setting up the Framework

```java
//Yup it's that easy!
m_felix = new Felix(configMap);
m_felix.start();
```
public <S> S getService(Class<S> serviceClass) {
    ServiceReference<S> ref = m_activator.getContext().getServiceReference(serviceClass);
    return m_activator.getContext().getService(ref);
}
public List<S> getServices(Class<S> serviceClass, String filter) {
    Collection<ServiceReference<S>> refCollection = null;
    try {
        refCollection = hostActivator.getContext().getServiceReferences(serviceClass, filter);
    } catch (InvalidSyntaxException e) {
        LOGGER.error("Invalid Syntax", e);
    }
    List<S> result = new ArrayList<S>();
    if (refCollection == null) {
        throw new ModuleException("No services References Could be found for the given class");
    }
    for (ServiceReference<S> currentRef : refCollection) {
        result.add(hostActivator.getContext().getService(currentRef));
    }
    return result;
}
DEMO!
Summary

- Modularity is often an afterthought
- Some projects have difficulty being fully OSGi
- Embedding OSGi can provide many of the same benefits
References

- Adding Modularity Afterwards with Embedded OSGi (Talk and Code)
- Felix Embedded Documentation
Bob Paulin

@bobpaulin/bob@bobpaulin.com/http://bobpaulin.com