Liberate your Components with OSGi Services

Graham Charters, IBM
<table>
<thead>
<tr>
<th>Level</th>
<th>Name</th>
<th>Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>AdHoc</td>
<td>Nothing</td>
</tr>
<tr>
<td>2</td>
<td>Modules</td>
<td>Formal identity, <em>decoupled from artifact</em></td>
</tr>
<tr>
<td>3</td>
<td>Modularity</td>
<td>Formal module contracts, <em>decoupled from identity</em></td>
</tr>
<tr>
<td>4</td>
<td>Loose-Coupling</td>
<td>Services, semantic versioning, <em>decoupled from implementation</em></td>
</tr>
<tr>
<td>5</td>
<td>Devolution</td>
<td>Modularity-aware repositories, collaboration, <em>decoupled from ownership</em></td>
</tr>
<tr>
<td>6</td>
<td>Dynamism</td>
<td>Life-cycle awareness and independence, <em>decoupled from time</em></td>
</tr>
</tbody>
</table>
Componentized Build

- Components have identity and version
- Components produce a jar

impl depends on interfaces at build time

Factory in interfaces uses Class.forName to load impl at runtime
Component Based Runtime

- A Java Bean
- Implements a specific interface
- Init/start/stop/destroy phases
- Started in a specified order
- Makes use of:
  - `Class.getResource()`
  - `Class.forName()`
Move to OSGi

- Learning curve
- Compatibility constraints

Option 1

Jar A
Jar B
Jar C
Jar D

Option 2

Content of Jars A - D

© 2013 IBM Corporation
Moving Forward Five Years

- More bundles
- Smaller bundles
- Packages split across bundles a real issue
- Limits on how much processing can be deferred
- Too much centralisation of control
- Removed some ill-fitting technologies
  - EMF
  - Update Provisioner
Issues with Approach

- All bundles are singletons
- Support outside OSGi framework required
- No notification support
- Too many inefficient APIs
- Data read up front during initialization
- Many different extension patterns
Why Services?

- Loose-coupling
- Lifecycle
- Multiple implementations
- Deferred Initialization
- Metadata driven service frameworks
Declarative Services

- XML Service Registration/Injection Framework
- Defines components
- Components are services
- Config Admin integration
- Setup (largely ☺) synchronous with bundle activation
- Annotations/bnd/xml

```
@Component
public class DiscountCalculatorImpl implements DiscountCalculator {

    private Logger logger;

    @Reference(policy=DYNAMIC,
                policyOption=GREEDY,
                cardinality=MANDATORY)
    void setLog( LogService log) { ... }
    void unsetLog( LogService log) { ... }
    void updatedLog( Map<String,?> ref) { ... }
}

@Component
class DiscountCalculatorImpl {
    private Logger logger;

    @Reference(policy=DYNAMIC,
                policyOption=GREEDY,
                cardinality=MANDATORY)
    void setLog( LogService log) { ... }
    void unsetLog( LogService log) { ... }
    void updatedLog( Map<String,?> ref) { ... }
}
```

```
<component name="DiscountCalculator" ...>
    <implementation
        class="com.acme.utils.impl.DiscountCalculatorImpl"/>
    <service>
        <provide
            interface="com.acme.utils.DiscountCalculator"/>
    </service>
    <reference name="Log"
        interface="org.osgi.service.log.LogService"
        bind="setLog" unbind="unsetLog"/>
</component>
```
XML Component Injection Framework

Inspired by Spring

Defines components
  – Beans
  – Services
  – References

Allows non-service components

Extensible

Asynchronous to bundle activation

Service Damping

5 minutes to die

```xml
<blueprint>
  ...
  <service id="DiscountCalculatorBeanService"
          ref="DiscountCalculatorBean"
          interface="com.acme.utils.DiscountCalculator" />
  <bean id="DiscountCalculatorBean"
       class="com.acme.utils.impl.DiscountCalculatorImpl">
    <property name="logger" ref="loggerService" />
  </bean>
  <reference id="loggerService"
            interface="com.acme.utils.Logger" />

</blueprint>
```
Liberty Profile 8.5.0

- New Server Kernel
- Mostly the same code
- Exploit standards where possible
- Be dynamic, lightweight, lazy

Backed by simple XML. “schema defined in metatype

Feature Manager

felix scr 1.6.1  equinox metatype 1.2.0  Config Admin R4.2

equinox framework 3.7.2
It’s not just about internal engineering

- Dynamically
- Composable
- Extensible
- Custom Profiles
Conclusion

- We had some pain along the way
  - DS thread-safety issue
  - Learning curve
  - Mindset change

- But, Services are
  - Powerful
  - Flexible
  - Lightweight
  - Lazy

- Loose-coupling

- Totally worth it