Liberate your components with OSGi services
One products journey through the Modularity Maturity Model

Alasdair Nottingham (not@uk.ibm.com)
WebSphere Application Server V8.5 Liberty Profile Development Lead
## Modularity Maturity Model Summary

<table>
<thead>
<tr>
<th>Level</th>
<th>Name</th>
<th>Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Ad Hoc</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, governance, <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.getResources()`
  - `Class.forName()`
Move to “OSGi”

Option 1

Jar A  Jar C
Jar B  Jar D

Option 2

Content of Jar A-D
Moving Forward Five Years

- More bundles
- Smaller bundles
- Packages split across bundles a real issue
- Limits on how much processing can be deferred
- To much centralisation of control
- Remove some non-standard eclipse technology
  - 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
Liberty Profile

- 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
Why services?

- OSGi Primitive
- Peer to peer
- Lifecycle
- Deferred Initialization
- Metadata driven service frameworks
Declarative Services

- XML Service Registration/Injection Framework
- Defines components
- Components are services.
- Config Admin integration
- Setup synchronously with bundle activation
- Annotations/bnd/xml
Blueprint

- XML Component Injection Framework
- Based on Spring
- Defines components
  - Beans
  - Services
  - References
- Allows non-service components
- Extensible
- Asynchronous to bundle activation
- Service Damping
- 5 minutes to die
When is optional truly optional?

Web Container

- transactions
- naming
- security

Optional, multiple, dynamic
Intermediaries

Web Container

Archive

Doesn't know about specific converters

Web Metadata
Ear Metadata
OSGi Metadata

Optional, multiple, dynamic

Needs an ArchiveManager, know the converters it cares about
Mixing Services and Static API calls

- Client
- Static Class
- Service A
- Service B
- Service C
When is a Bundle ACTIVE?

- Services come and go
- Dynamically
- So what does ACTIVE really mean?
- Are we ready for the ready service?
Shutdown

- Synchronous for mandatory services
- Asynchronous for optional services
Service versioning

Service Client

Service interface Bundle

Bundle A version 1.0.0

Bundle A version 1.0.1

which service to use?
Conclusion

- Services are
  - Powerful
  - Flexible
  - Lightweight
  - Lazy
  - Peer to Peer

- Complicated
- Totally worth it
Questions?