Best Practices for (Enterprise) OSGi applications
Who is Tim Ward?
- @TimothyWard

- Java Consultant
- 5 years at IBM developing WebSphere Application Server
- PMC member of the Apache Aries project
  - Particularly interested in Bytecode weaving/generation, JPA, EJBs, Blueprint Dependency injection, Declarative qualities of service (e.g. transactions)
- Regular conference speaker
  - JAX London, EclipseCon, Devoxx, Jazoon, OSGi Community Event
- Author of Enterprise OSGi in Action
  - Early access available at http://www.manning.com/cummins
  - All chapters complete
  - See the Manning booth for discounts on this and many other books
Agenda

What is “Enterprise OSGi” and why do I need it?

How can I use OSGi in my Applications?

How *should* I use OSGi in my Applications?

Where can I learn more about OSGi?
What is “Enterprise OSGi” and why do I need it?

- OSGi is a mature technology with a broad range of adoption
  - Eclipse!
  - Embedded systems
  - Home automation
  - Java EE Application Servers

- Enterprise OSGi is much newer (First release 2010)
  - Primary focus to improve OSGi’s support for enterprise tools
  - Widely available in Open Source and Commercial servers
What is “Enterprise OSGi” and why do I need it? (2)

- So simply put Enterprise OSGi is just OSGi applied to “Enterprise” Applications
  - OSGi Web applications
  - Using databases from an OSGi framework
  - Managed Transactions for OSGi bundles
  - Remoting Services…

- But isn’t this what Java EE is for?
  - Why is OSGi helpful?
OSGi is used for many reasons, but a primary motivation is modularity.

Big systems are hard to maintain and understand because of the relationships between components:

- Big applications are just as complicated as servers (and usually have more external dependencies!)
Why else do I need OSGi?

• Have you ever found that you need to use a library class, but it depends on another version of a library you were already using?
  • Java has a flat classpath, so you can only have one version of the class
  • If you can’t change the code you can be forced into using brittle combinations of point releases

• OSGi has a classloader graph:
  • It all just works!
How can I use OSGi in my Applications?

- Lots of Application runtimes offer some support for OSGi applications
  - WebSphere, Glassfish, Jboss, Geronimo, Karaf, Virgo, Aries…
- Most require little more than packaging your application as OSGi bundles

**OSGi Bundle**

**Manifest**

- **Bundle-ManifestVersion**: 2
- **Bundle-SymbolicName**: com.acme.my.bundle
- **Bundle-Version**: 1.0.0
- **Import-Package**: com.acme.useful.package, com.acme.another.useful.package
- **Export-Package**: com.acme.api.package
In terms of scope OSGi bundles are like JARs with better isolation

- This is good, but how many Enterprise Applications are built as a single JAR?

The Java EE EAR exists to support multi-module applications

- Even WAR files have built in support for library JARs

For a long time OSGi had no scope beyond the bundle

- Parallel solutions exist in Eclipse, Apache and several commercial servers
- A unified model is offered by OSGi Subsystems
How *should* I use OSGi in my Applications?

Best practices for all OSGi applications
How *should* I use OSGi in my Applications?

### 1. Bundle Hygiene

- Well designed Object Oriented code exhibits good modular properties
  - Simple reuse and ability to switch implementation

- These properties rely on classes being Cohesive and loosely coupled

- From a modularity perspective OSGi bundles very similar
  - To work well an OSGi Bundle should be cohesive and loosely coupled

- The bundle manifest is an excellent guide to how well designed the bundle actually is
How *should* I use OSGi in my Applications?

1 a) Avoid Tight Coupling

- Java developers learn to recognise tightly coupled code
  - Casting to implementation
  - Relying on side-effects and leftover state

- OSGi has similar warning signs
  - We saw how code can be imported by `Import-Package`
  - OSGi also offers `Require-Bundle`

- Require-Bundle is a bit like casting to an implementation type
  - You don’t just care about API, but also where it comes from!
How *should* I use OSGi in my Applications?

1 b) Do enough to be Cohesive

- Splitting behaviour across multiple Objects makes an API hard to use
  - The same is true of packages in OSGi

- A split package is one that exists two or more bundles but contains different classes in each
  - The OSGi classloader allows packages to come from exactly one bundle
How *should* I use OSGi in my Applications?
1 c) Don’t do too much in your bundle

- Doing too much in a class is as bad as doing too little
  - It’s hard to use an API with too many methods and arguments
  - It adds overhead and hurts performance
  - It’s hard to maintain

- Bundles can suffer from the same problem
  - Huge numbers of dependencies (Import-Package or Require-Bundle)
  - Lots of exported packages (which one do I use?!?)

- If a manifest can be measured in megabytes you’re doing it wrong!
How should I use OSGi in my Applications?

2. Version Everything

- You’ve seen that bundles can have a version
  - Package Exports can be versioned too
  - Imports can declare a range of accepted versions

- Versioning properly makes bundles less brittle and easier to reuse
  - Semantic versioning of packages allows clients to declare what function they require
    - Major version changes indicate that clients might be broken
    - Minor version changes indicate backward compatible updates
    - Micro version changes are for bug fixes

- Unversioned packages are like a box of chocolates…
How *should* I use OSGi in my Applications?

2. Version *Everything* (2)

- We saw how split packages can break client bundles
  - If they are versioned properly then the client can make a choice
How *should* I use OSGi in my Applications?

3 a) Use Services for looser coupling

- Java lacks a satisfactory way to get implementation Objects
  - Using new introduces tight coupling
  - Using a factory is better, but still couples you to the factory!
- OSGi has a service registry that bundles can use to collaborate
  - Services are registered using their API, so clients don’t need to construct them!
- Using services makes it much easier to reuse and swap bundles
3 b) Make services substitutable

- Sharing services relies on your bundles using the same version of the API
  - In fact it relies on you both getting the API from the same bundle!

- If the same API is available from multiple bundles then the client might see a different one to the service provider…

- API works best when it is substitutable
  - Bundles that export API should import it too – this allows more sharing
How *should* I use OSGi in my Applications?

Best practices for *Enterprise* OSGi applications
How *should* I use OSGi in my Applications?

4. Don’t do it all yourself!

- OSGi is a very powerful environment, but it can be hard to use
  - Many constructs are very low level
  - This is great for embedded systems, but not always for enterprise apps!

- The Enterprise Specifications offer a number of helpful tools
  - Dependency injection frameworks
  - Data Access Services…

- Make sure you use them!
How *should* I use OSGi in my Applications?

4 a) Accessing services

- **Using an OSGi service *properly* is hard because they are dynamic**

```java
private BundleContext ctx;

private AtomicReference<LogService> ls = new AtomicReference<LogService>();
private AtomicReference<ServiceReference> lr = new AtomicReference<ServiceReference>();

public void start(BundleContext ctx) throws InvalidSyntaxException {
    this.ctx = ctx;
    ctx.addServiceListener(this, "(objectClass=org.osgi.service.log.LogService"));
    ServiceReference ref = ctx.getServiceReference(LogService.class.getName());
    if (ref != null) {
        ls.set((LogService) ctx.getService(ref));
        lr.set(ref);
    }
}

@Override
public void serviceChanged(ServiceEvent event) {
    ServiceReference ref = event.getServiceReference();

    if (ls.get() == null && event.getType() == ServiceEvent.REGISTERED) {
        ls.set((LogService) ctx.getService(ref));
    } else if (ls.get() != null && event.getType() == ServiceEvent.UNREGISTERING &&
              ref == lr.get()) {
        ref = ctx.getServiceReference(LogService.class.getName());
        if (ref != null) {
            ls.set((LogService) ctx.getService(ref));
            lr.set(ref);
        }
    }
}
```
How should I use OSGi in my Applications?

4 a) Accessing services (2)

- There are several OSGi dependency injection containers that make using services much easier
  - Blueprint and Declarative Services are both OSGi standards

- Declarative Services is very lightweight
  - Great for systems with simple wirings
  - No damping of services (A good and bad thing!)

- Blueprint offers a Spring-like programming model
  - Easy to set up and manage large injection graphs
  - Service damping means beans are protected from the service lifecycle
How *should* I use OSGi in my Applications?

4 a) Accessing services (3)

Sample blueprint consuming a service:

```xml
<blueprint>
  <bean id="myBean" class="org.acme.impl.MyBean">
    <property name="logService" ref="logService" />
  </bean>
  <reference id="logService"
    interface="org.osgi.service.log.LogService" />
</blueprint>
```

Sample Declarative Services consuming a service:

```xml
<component name="myBean">
  <implementation class="org.acme.impl.MyBean" />
  <reference bind="setLogService" cardinality="1..1"
    interface="org.osgi.service.log.LogService"
    policy="static" unbind="unsetLogService"
    name="logService" />
</component>
```
How *should* I use OSGi in my Applications?

4 b) Accessing data

- JDBC and JPA are commonly used to access data
  - Both rely on static factories and Classpath visibility to work

- Trying to use traditional access patterns leads to unpleasant hacks
  - There are Standard ways to get hold of these things in OSGi

- The JDBC service uses DataSourceFactory services to create DataSources
  - Your server may register managed DataSource services too

- The JPA service also uses the service registry to provide EntityManagerFactory Objects
Traditionally OSGi applications use the HttpSession to register Servlets:

- This is good if you have one or two servlets, but not for big web apps

The Enterprise Specification defines Web Application Bundles:

- Essentially they are WARs with OSGi metadata

WABs allow you to reuse tools and expertise when moving to OSGi:

- Many web frameworks are OSGi enabled too
- A very easy way to begin migrating to OSGi
Summary
Things to remember

- OSGi isn’t as hard as you’ve been led to believe!
  - But it isn’t magic either, you need to use what it gives you

1. Keep your bundles tidy and well defined
   i. Spaghetti bundles are just as bad as spaghetti code!

2. Use semantic versioning to keep control of your dependencies

3. Use the service registry to communicate between bundles in a simple, decoupled way

4. Use the Enterprise specifications to avoid writing huge amounts of boilerplate in your applications
Useful Resources

• The OSGi specifications are available at http://www.osgi.org
• Apache Aries for implementations http://aries.apache.org/
• Manning have several good OSGi books
  • **Enterprise OSGi in Action** – Get up and running with Web Apps, Transactions, JPA, Remoting, IDEs and build tools
  • **OSGi in Action** – Great examples and coverage of core OSGi and compendium services
  • **OSGi in Depth** – Detailed coverage of architectural patterns for OSGi
• If you go to the Manning stand there are big EclipseCon discounts
• OSGi Articles available at http://www.developerworks.com