

An introduction to OSGi

Dave Snowdon

Ecube Ltd

dave.snowdon@gmail.com

Contents

- What's OSGi
- Anatomy of a bundle
- Basic examples
- Spring dynamic modules
- Serving web pages using OSGi

Who? Why?

- Just another web developer
 - Not buzzword compliant
- Have a chance to update Ecube's architecture and want to do it "right" (or at least "better")
- The OSGi advertising sounds good, wanted to learn more

Questions to ask

- What do we get?
- What does it cost?
 - How much do we need to change our code?
 - How much bloat does it add?
 - Any efficiency concerns?

What is OSGi?

- OSGi technology is the dynamic module system for Java™. The OSGi Service Platform provides functionality to Java that makes Java the premier environment for software integration and thus for development. Java provides the portability that is required to support products on many different platforms. The OSGi technology provides the standardized primitives that allow applications to be constructed from small, reusable and collaborative components. These components can be composed into an application and deployed.
- The OSGi Service Platform provides the functions to change the composition dynamically on the device of a variety of networks, without requiring restarts. To minimize the coupling, as well as make these couplings managed, the OSGi technology provides a service-oriented architecture that enables these components to dynamically discover each other for collaboration. The OSGi Alliance has developed many standard component interfaces for common functions like HTTP servers, configuration, logging, security, user administration, XML and many more. Plug-compatible implementations of these components can be obtained from different vendors with different optimizations and costs. However, service interfaces can also be developed on a proprietary basis.

OK, so what is OSGi really?

Specification for:

- Packaging bundles (modules)
- Runtime
 - Class loading policies
 - Can express dependences between bundles
 - Lifecycle: Install, start, stop, upgrade bundles in running system
 - Service registry
 - Management console

Open Source Implementations

- Eclipse Equinox
 - If you use Eclipse 3, then you're already using OSGi
 - Equinox is now the way Eclipse plugins are packaged
 - Eclipse support for OSGi bundle development
- Knoplerfish
- Apache Felix

Anatomy of a bundle

JAR file

META-INF / MANIFEST.MF

STUFF

Anatomy of a bundle

- Just jar files with a META-INF/MANIFEST.MF
 - But jar files are not necessarily bundles!
- Bundles can act as ordinary jar files
- Has own class path
- Can declare dependencies on native code
- Dependencies modelled by java packages, not bundles
- Can export packages to other bundles
- Can import packages from other bundles

META-INF / MANIFEST.MF

```
Bundle-ManifestVersion: 2
Bundle-Name: OSGI Demo 1
Bundle-SymbolicName:
    uk.co.ecube.osgi.demo1
Bundle-Version: 1.0.0
Bundle-Activator:
    uk.co.ecube.osgi.demo1.Activator
Export-Package: bar; version="1.0.0"
Import-Package: foo;
    version="[1.0.0,1.5.0)"
```

Will this work?

```
package uk.co.ecube.foo;  
public class Thing {  
    ....  
}
```

```
package uk.co.ecube.bar;  
import uk.co.ecube.foo.Thing;  
public class ThingUser {  
    public void doSomething() {  
        Thing thingy = new Thing();  
        ...  
    }  
}
```

Activator

```
package uk.co.ecube.osgi.demo1;

import org.osgi.framework.BundleActivator;
import org.osgi.framework.BundleContext;

public class Activator implements BundleActivator {

    public void start(BundleContext context) throws Exception {
        System.out.println("Hello World!!");
    }

    public void stop(BundleContext context) throws Exception {
        System.out.println("Goodbye Cruel World!!");
    }
}
```

Making a service available (1)

```
public class Activator implements BundleActivator {  
  
    ServiceRegistration buzzwordServiceRegistration;  
  
    public void start(BundleContext context) throws Exception {  
        BuzzwordService buzzwordService = new BuzzwordImpl();  
  
        buzzwordServiceRegistration =  
            context.registerService(BuzzwordService.class.getName(),  
                                    buzzwordService, new Hashtable());  
    }  
  
    public void stop(BundleContext context) throws Exception {  
        buzzwordServiceRegistration.unregister();  
    }  
}
```

Service consumer (1)

```
public class Activator implements BundleActivator {
    ServiceReference _buzzwordServiceReference = null;

    public void start(BundleContext context) throws Exception {
        _buzzwordServiceReference =
            context.getServiceReference(
                BuzzwordService.class.getName());

        BuzzwordService buzzwordService = (BuzzwordService)
            context.getService(_buzzwordServiceReference);

        System.out.println("I think you need some “
            +buzzwordService.buzzword());
    }

    public void stop(BundleContext context) throws Exception {
        context.ungetService(_buzzwordServiceReference);
    }
}
```

Bundle status

- *Not in the list* - If your bundle isn't in the list, then OSGi doesn't know anything about it.
- **INSTALLED** - This means that OSGi knows about your bundle but there is something wrong and it couldn't resolve.
- **RESOLVED** - All dependencies resolved but bundle not running (could be a startup problem).
- **<<lazy>>** - This means your bundle is resolved and is marked to be lazy started. Everything should be ok.
- **ACTIVE** - your bundle is resolved and has been started, everything should be working as planned.

Console

Some sample commands:

- `ss` - shows the installed bundles, their IDs and their status
- `install <JAR>` - loads a jar file into the container
- `diag <ID>` - diagnostic on the specified bundle. ie information about any missing imported packages
- `start <ID>` - start a bundle
- `stop <ID>` - stop a bundle
- `exit`

Demo #1

- Consumer 1
- Service 1
- Simple service & consumer
- OSGi console

Service factories

- In previous example the service was a single object
- What if we want to create a new object for each consume?
 - Use a service factory

Service factory

```
public class BuzzwordServiceFactory implements ServiceFactory {  
  
    public Object getService(Bundle bundle,  
                             ServiceRegistration registration) {  
  
        BuzzwordService buzzwordService = new BuzzwordImpl();  
        return buzzwordService;  
    }  
  
    public void ungetService(Bundle bundle,  
                             ServiceRegistration registration,  
                             Object service) {  
        // whatever we need to do to shut down the service object  
    }  
}
```

Activator for a factory

```
public class FactoryActivator implements BundleActivator {
    ServiceRegistration buzzwordServiceRegistration;

    public void start(BundleContext context) throws Exception {
        BuzzwordServiceFactory buzzwordServiceFactory =
            new BuzzwordServiceFactory();

        buzzwordServiceRegistration =
            context.registerService(BuzzwordService.class.getName(),
            buzzwordServiceFactory, null);
    }

    public void stop(BundleContext context) throws Exception {
        buzzwordServiceRegistration.unregister();
    }
}
```

Tracking services

- Suppose we want to know when a service is available and when it becomes unavailable?
 - Can use a service tracker to get notifications

Service tracker

```
public class BuzzwordServiceTracker extends ServiceTracker {  
  
    public BuzzwordServiceTracker(BundleContext context) {  
        super(context, BuzzwordService.class.getName(),null);  
    }  
  
    public Object addingService(ServiceReference reference) {  
        return super.addingService(reference);  
    }  
  
    public void removedService(ServiceReference reference,  
                               Object service) {  
        super.removedService(reference, service);  
    }  
}
```

Activator using a tracker

```
public class TrackerActivator implements BundleActivator {

    BuzzwordServiceTracker buzzwordServiceTracker;

    public void start(BundleContext context) throws Exception {
        buzzwordServiceTracker = new BuzzwordServiceTracker(context);
        buzzwordServiceTracker.open();
        BuzzwordService buzzwordService =
            (BuzzwordService) buzzwordServiceTracker.getService();

        if (null != buzzwordService) {
            System.out.println("I think you need some “
                +buzzwordService.buzzword());
        }
    }

    public void stop(BundleContext context) throws Exception {
        buzzwordServiceTracker.close();
    }
}
```

Demo #2

- Consumer 2
- Service 2
- Factories and trackers

Spring DM

- Programmatically exporting and locating services could get tedious very quickly
- Spring DM can locate services for us
- As far as we are concerned Spring DM wiring is much like a normal Spring application.
- Works with “Spring powered” bundles
 - XML files in META-INF / spring

Look Ma, no activator!

META-INF/MANIFEST.MF

Manifest-Version: 1.0

Bundle-ManifestVersion: 2

Bundle-Name: Osg1_spring1 Plug-in

Bundle-SymbolicName:

uk.co.ecube.osgi.spring1

Bundle-Version: 1.0.0

Import-Package:

org.osgi.framework;version="1.3.0"

META-INF/spring/helloworld.xml

```
<?xml version="1.0" encoding="UTF-8"?>
  <beans xmlns=
http://www.springframework.org/schema/beans
    xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
    xsi:schemaLocation="http://www.springframework.org/sche
    ma/beans
    http://www.springframework.org/schema/beans/spring-
    beans.xsd">

    <bean name="hello"
      class="uk.co.ecube.osgi.spring1.HelloWorld"
      init-method="start"
      destroy-method="stop" />
  </beans>
```

Spring DM service example

- We export a simple service
- Client relies on Spring to get service reference via DI
- Client calls getUsers() on the service

```
public interface MembershipService {  
    public List<User> getUsers();  
    public User getUser(String id);  
    public void saveUser(User user);  
    public void deleteUser(String id);  
}
```

Config

Service

- META-INF / spring
 - membership-service.xml
 - membership-osgi.xml

Client

- META-INF / spring
 - Client.xml
 - Client-osgi.xml

Declare our spring service

META-INF / spring / service.xml

...

```
<bean name="membershipService"
```

```
    class="uk.co.ecube.osgi.membership.impl.  
    MembershipServiceImpl">
```

```
</bean>
```

...

Export spring service as OSGi service

```
<osgi:service
  id="membershipOSGiService"
  ref="membershipService"

  interface="uk.co.ecube.osgi.membership.MembershipService">
</osgi:service>
```

Import OSGi service

...

```
<osgi:reference
```

```
  id="membershipService"
```

```
  interface="uk.co.ecube.osgi.membership.  
  MembershipService"/>
```

...

Use the imported service

```
<bean name="hello"  
  class="uk.co.ecube.osgi.ui.HelloWorld"  
  init-method="start"  
  destroy-method="stop" >  
  <property  
    name="membershipService"  
    ref="membershipService"/>  
</bean>
```

Demo #3

- Spring service 1
- Spring consumer 1
- Shows how we can rely on spring to handle wiring of applications composed of different modules

OSGi and the web

- 2 approaches for implementing web applications using OSGi
 - Embed a servlet container in the OSGi container
 - Jetty bundle for equinox
 - Embed an OSGi container in a web application running in a servlet container
 - Servlet (bridge.war) that runs equinox as a web app
- Luckily both approaches look the same from the application point of view

Publishing web resources

- OSGi container has `HttpService`
- Bundles wanting to handle HTTP requests registers URLs it can handle with `HttpService`
- URLs can be servlets or static resources
- Programmatic: Can invoke methods on `HttpService` to add URLs
- Declarative: Declare URLs in `plugin.xml`

Programmatic approach using service tracker

```
public Object addingService(ServiceReference reference) {  
  
    HttpService httpService = (HttpService)  
    context.getService(reference);  
    try {  
        httpService.registerResources(  
            "/helloworld.html", "/helloworld.html", null);  
        httpService.registerServlet("/helloworld",  
            new HelloWorldServlet(), null, null);  
    } catch (Exception e) { ... }  
    return httpService;  
}
```

Declarative approach using plugin.xml (1)

```
<extension
  id="helloResource"
  point="org.eclipse.equinox.http.registry.res
  ources">
  <resource
    alias="/decl/helloworld.html"
    base-name="/helloworld.html" />
</extension>
```

Declarative approach using plugin.xml (2)

```
<extension
  id="helloServlet"
  point="org.eclipse.equinox.http.registry.servlets"
  >
  <servlet
    alias="/decl/helloworld"

    class="com.javaworld.sample.osgi.web.webapp.
    HelloWorldServlet">
  </servlet>
</extension>
```

Spring DM & web

- Tomcat & Jetty supported
- Watches for .war installed in the container
- Integrates with Tomcat Jasper 2 engine to provide support for JSP
- Integrates with Spring MVC
 - `OsgiBundleXmlWebApplicationContext` replaces `XmlWebApplicationContext`

Bloat check

- Equinox
 - jars ~1Mb
 - ~ 465 classes

Alternatives

- Phil Zoio's Impala framework
 - Dynamic modules for Spring applications
 - Presented, by Phil, at the JAVAWUG BOF 37 (20th May 2008)
 - <http://code.google.com/p/impala/>

Conclusions

- OSGi can help application development
 - Enforces module boundaries
 - Live updates
- Writing code to connect up services could become tedious
- For Spring developers Spring DM seems like a relatively painless transition

Links

- OSGi : www.osgi.org
- Equinox: <http://www.eclipse.org/equinox/>
- Server side Equinox: <http://www.eclipse.org/equinox/server/>
- Apache Felix: <http://felix.apache.org/site/index.html>
- Knoplerfish: <http://www.knoplerfish.org/>
- Spring DM: <http://www.springframework.org/osgi/>
- http://wiki.eclipse.org/Where_Is_My_Bundle
- <http://www.javaworld.com/javaworld/jw-03-2008/jw-03-osgi1.html>
- <http://www.javaworld.com/javaworld/jw-04-2008/jw-04-osgi2.html>
- <http://www.javaworld.com/javaworld/jw-06-2008/jw-06-osgi3.html>
- OSGi+Jetty tips:
<http://docs.codehaus.org/display/JETTY/OSGi+Tips>
- Impala: <http://code.google.com/p/impala/>