

OSGi Alliance Community Event

**Everything can be a bundle –
Making OSGi bundles of Java legacy code**

Erik Wistrand
wistrand@makewave.com

Gunnar Ekolin
ekolin@makewave.com



What's the problem?

- Lots of Java code has been written taking stuff for granted
 - Just one classloader
 - `System.exit()` is OK to call
 - A file system is present
 - A static `main()` method is used for start-up
 - ...

...this leads to classloading problems, premature exit of the entire framework and various internal bugs



What can be done?

The Knopflerfish OSGi framework contains code that can

- Start jar files with only a static main method
- Automatically import/export packages
- Patch code that uses “bad” methods regarding classloading etc.



Installing a non-bundle JAR-file

- The Auto-Manifest feature of Knopflerfish makes it possible to take any jar file and install it as if it were bundle
 - All contained packages will be exported
 - `Dynamic-Import: *` used to get access to shared packages.



Auto-Manifest - Configuration

The file `automanifest.props`

```
1.match.filter=(location=*hello_world-app*) ^  
1.export.filter=(pkg=*) ^  
1.export.file.filter=(file=*.class) ^  
1.header.DynamicImport-Package=*  
1.header.Import-Package=[remove]  
1.header.Export-Package=[autoexport]
```



Auto-Manifest - Launch

```
java
-Dorg.knopflerfish.framework.automanifest=true
-Dorg.knopflerfish.framework.automanifest.config=
  file:automanifest.props
-Dorg.knopflerfish.framework.debug.automanifest=
  true
-jar framework.jar
```



Demo: Hello World

- Install the Hello World application `hello_world-app-1.0.0.jar`
- The demos are available for download from <http://www.knopflerfish.org/demos/knopflerfish-osgi-berlin-2008.zip>



Main-Class as Activator - Launch

```
java
-Dorg.knopflerfish.framework.automanifest=true
-Dorg.knopflerfish.framework.automanifest.config=
  file:automanifest.props
-Dorg.knopflerfish.framework.debug.automanifest=
  true
-Dorg.knopflerfish.framework.main.class.activation=
  file:jars/hello_world-app/hello_world-app-1.0.0.jar
-jar framework.jar
```



System.exit()

- An application normally calls `System.exit(0)` to terminate.
 - This will **kill** the **entire** OSGi framework.
- Common workaround:
 - Modify the application source to be OSGi aware and recompile.
 - Impractical.
 - License issues.
 - Source may not be available.



A new approach – automatic byte code modification

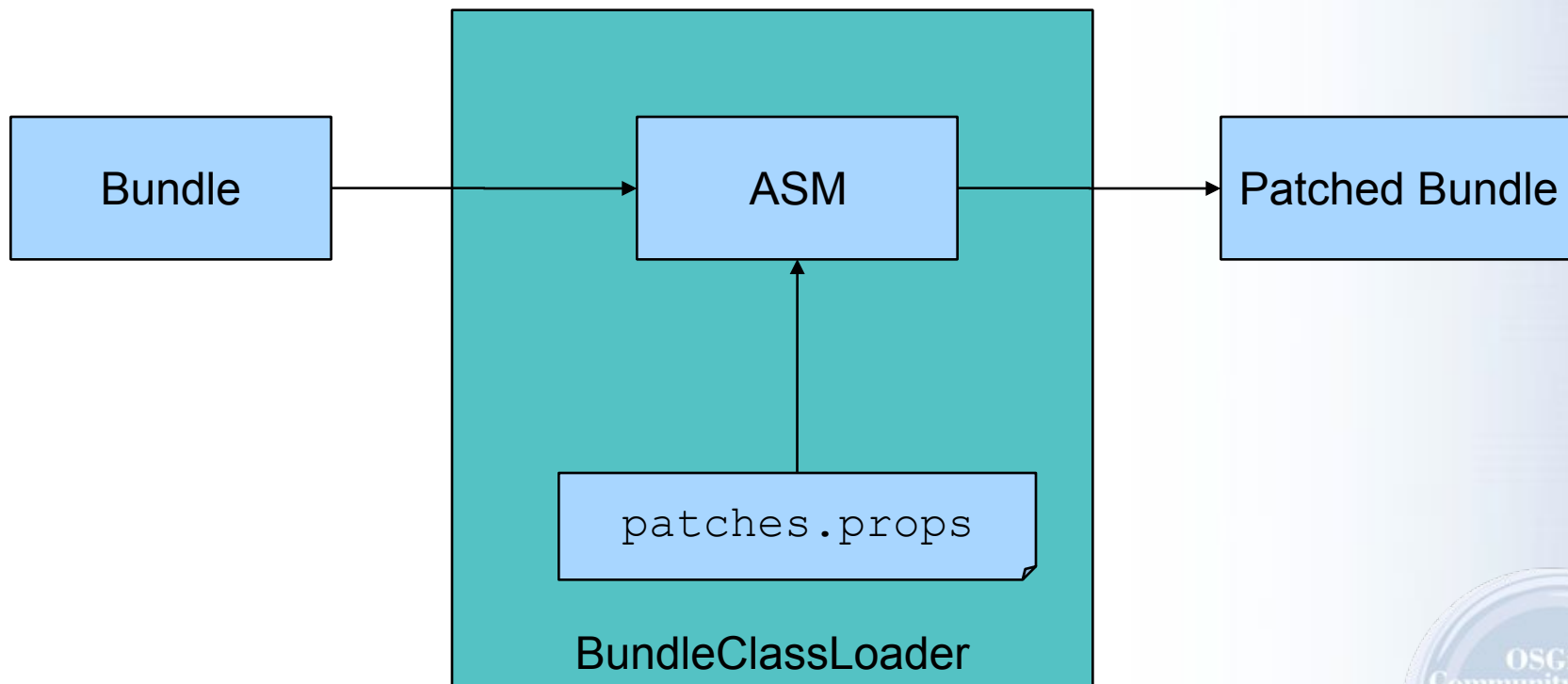
Since all bundle classes are loaded via the OSGi framework, the framework can modify the byte-code on the fly to use the correct classloader:

1. Load original byte code.
2. Find occurrences of “wrong” method calls.
3. Replace with call to “right” method.
4. Use the modified class.

(5. Profit!)



The Knopflerfish implementation



The Knopflerfish implementation

- Uses the ASM byte-code manipulation library.
- A configuration file can specify which bundles/classes should be modified.
- Bundles are automatically modified as they are loaded.

Yes, this is a poor man's **aspect oriented** framework.

btw, LDAP filters are the best thing since sliced bread. Yes, they are!



Default set of patches

- `java.lang.ClassLoader.getSystemClassLoader()` ↑
 - Returns the bundle classloader.
- `java.lang.System.exit(int)` ↑
 - **Calls** `BundleContext.stop()`, disabled by default.
- `java.lang.Class.forName(String)` ↑
 - First tries the default classloader, then the bundle's classloader.
- `java.lang.Class.forName(String, boolean, ClassLoader)` ↑

F

irst tries the specified classloader, then the bundle's classloader.



Patching - Launch

```
java  
-Dorg.knopflerfish.framework.patch=true  
-Dkf.patch.systemExitWrapper=true  
-cp framework.jar:asm-3.1.jar  
org.knopflerfish.framework.Main
```



Demo: Hello World

- Start the Hello World application and stop it by closing the window.
- Same thing again with default bundle patching enabled.

- The demos are available for download from

<http://www.knopflerfish.org/demos/knopflerfish-osgi-berlin-2008.zip>



Classloading problems

- Old, non-OSGi-aware libraries can easily be wrapped as an OSGi bundle.
 - Sounds easy, just create a manifest file and export/import the necessary stuff,

...but they may still run into classloading problems.

```
ClassNotFoundException:  
  Cannot find com.acme.ICustomer
```



Why ClassNotFoundException

- For the OSGi import/export mechanism to work, the bundle must use the Bundle classloader.
- However, many libraries uses the system classloader or the thread's classloader
 - Works great in standalone apps
 - Does **not work at all** in typical OSGi settings



```
package mypackage;
```

```
class Foo {
```

```
    Class c = ...getContextClassLoader().loadClass("mypackage.Bar");
```

```
    Class c = Foo.class.getClassloader().loadClass("mypackage.Bar");
```

Fails!

ok!

Bundle classloader

mypackage.Foo.class
mypackage.Bar.class

OSGi framework

System classloader

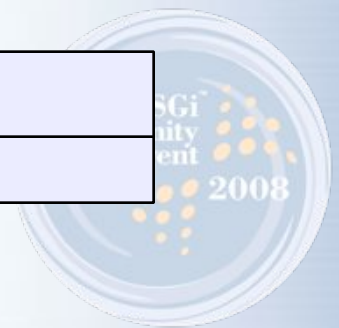
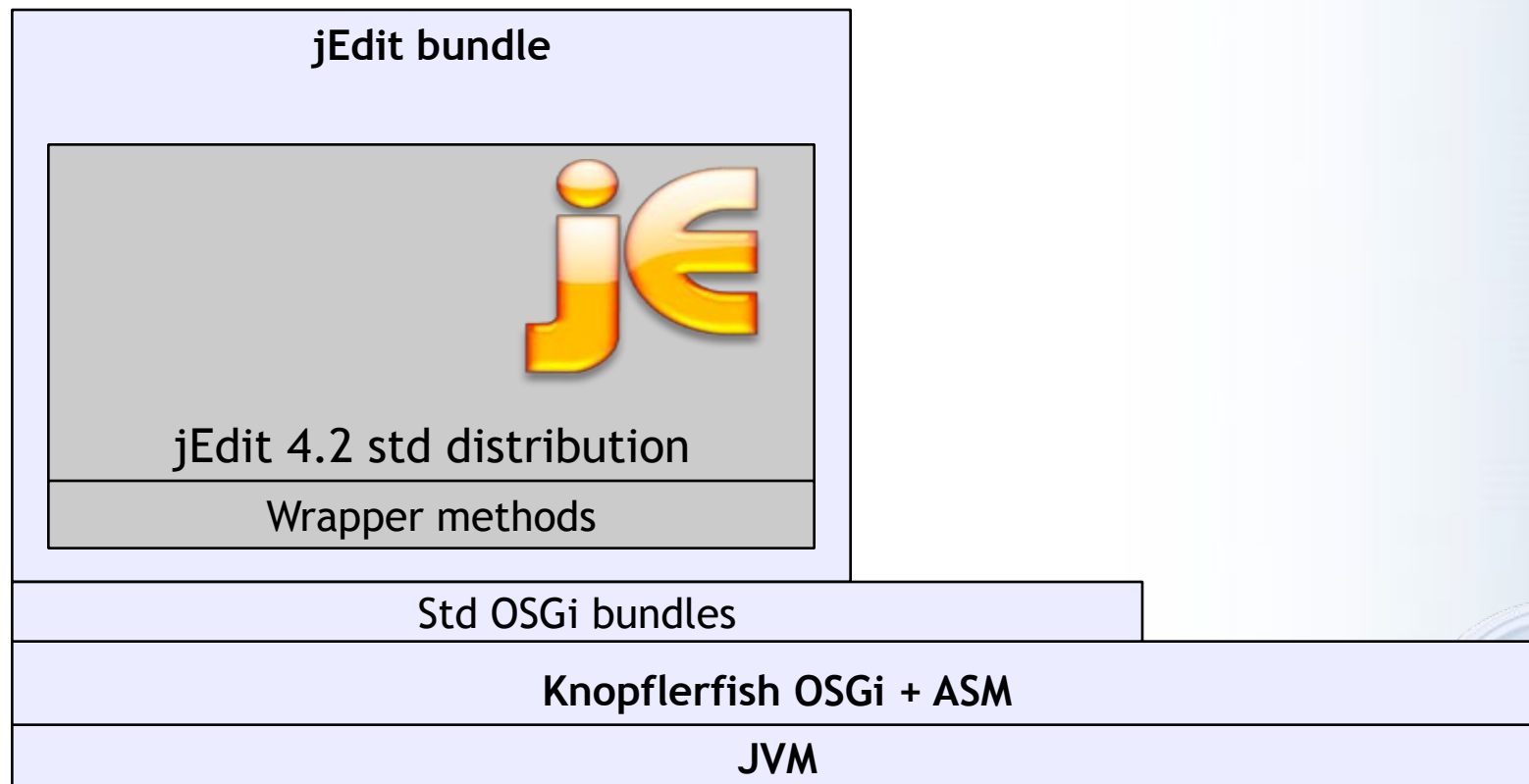
2008

Common workarounds to fix classloading

- Modifying the library source to be OSGi-aware and recompile
 - Impractical
 - License issues
- Wrapping each library call in code that sets the thread's context class loader
 - Tricky to find all cases
 - Boilerplate code is a Bad Thing
- Put all classes on the system class path!
 - Really just hurts



Demo – run jEdit as a bundle



Example patch

Let's patch all library calls to

```
Class.forName(String name,  
              boolean init,  
              ClassLoader cl) {
```



First, write the wrapper method

```
public static
  Class forName3Wrapper (String    name,
                        boolean    initialize,
                        ClassLoader cl,
                        long        bid,
                        Object      context)
  throws ClassNotFoundException
{
    // use bundle classloader instead
    return ...
}
```



Next, find signature strings

- Find the string signature of the method to be patched

```
java/lang/Class.forName(Ljava/lang/  
String;ZLjava/lang/ClassLoader;)Ljava/  
lang/Class;
```

- Find the string signature or the static wrapper method that should replace the above call

```
org/knopflerfish/framework/  
ClassPatcherWrappers.forName3Wrapper
```



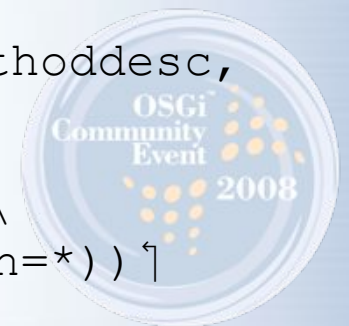
Create configuration file

```
patch.1.from= java/lang/Class.\
  forName(Ljava/lang/String;ZLjava/lang/ClassLoader;)\
  Ljava/lang/Class;

p
  at
    ch.1.to= org/knopflerfish/framework/ClassPatcherWrappers.\
    forName3Wrapper

patch.1.static=true
patch.1.active=true

# Optional LDAP-filter with keys: classname, methodname, methoddesc,
#   methodaccess, bid, location and all manifest headers.
patches.filter=(&(! (classname=org.knopflerfish.*))\
  (! (classname=org.osgi.*)) (location=*)) |
```



Specify Configuration File

- A global (default) patch configuration file can be specified as the value of the (system) property:

```
org.knopflerfish.framework.patch.configurl
```

- **Each bundle can specify a bundle specific patch configuration file via the manifest header:**

```
Bundle-ClassPatcher-Config:!/patches.props
```



...and launch

```
java \  
-Dorg.knopflerfish.framework.patch=true \  
-Dorg.knopflerfish.framework.patch.configurl=\  
  file:patches.props \  
-cp framework.jar:asm-3.1.jar \  
org.knopflerfish.framework.Main
```



Upcoming changes

- Today the patching engine is part of the framework.
- It will be moved out to a separate bundle using an API similar to

```
java.lang.instrumentation.ClassFileTransformer
```





June 10-11, 2008 Berlin, Germany

OSGi Alliance Community Event

Q&A

Gunnar Ekolin
ekolin@makewave.com



OSGi Alliance Community Event

**Everything can be a bundle –
Making OSGi bundles of Java legacy code**

Erik Wistrand
wistrand@makewave.com

Gunnar Ekolin
ekolin@makewave.com

