Scripting Languages in OSGi

ApacheCon Europe
Sinsheim, Germany 5th-8th November 2012

Frank Lyaruu
CTO Dexels
Project lead Navajo Framework
Amsterdam
www.dexels.com
Twitter: @lyaruu

DEXELS

Navajo Framework

- TSL
- XML based script language
- Compiled to Java
- Recently ported to OSGi


Scripting languages in OSGi

- The combination has a pretty bad reputation
- Different approach to dynamism

Scripting languages in OSGi

- Use any (?) scripting language
- Dynamically add, modify and remove scripts
- Expose it as an OSGi service
- Can declaratively depend on other services
- No external depedencies
- Behave like any other bundle

DEXELS

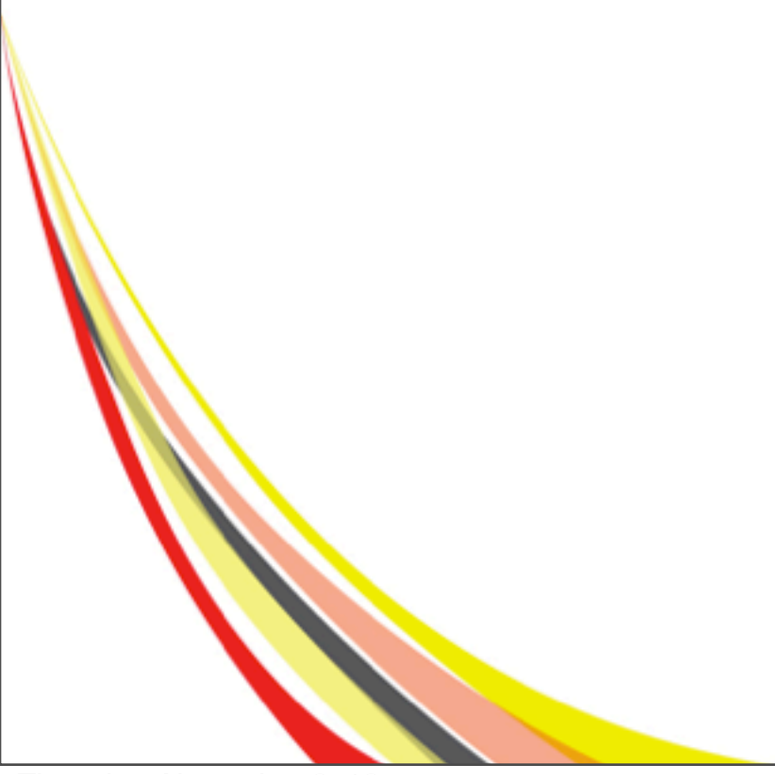
Advantages of scripting languages

- Dynamic. Changing code at runtime is completely normal
- No complex builds necessary
- Effective languages available for certain problems

Disadvantages of scripting languages

- Errors might reveal themselves only at runtime
- Limited modularity
- Performance can be a problem

Calling a script from Java



DEXELS

JSR-223 Scripting API

- More or less language independent
- Many supporting languages
- Some have a proprietary API
- Some have both

DEXELS

Define a java interface

In the simplest case:

```
package dexels.apachecon.api;

import java.io.IOException;

public interface Evaluator {
    public Object evaluate(Object o) throws ScriptException, IOException;
}
```

Using scripts in plain Java

```
package dexels.apachecon.plainjava;

import java.io.FileReader;

public class JsExample implements Evaluator {

    private final String scriptName;

    public JsExample(String name) {
        this.scriptName = name;
    }

    @Override
    public Object evaluate(Object o) throws ScriptException, IOException {
        final ScriptEngineManager factory = new ScriptEngineManager();
        final ScriptEngine engine = factory.getEngineByName("js");
        Bindings bindings = engine.getBindings(ScriptContext.ENGINE_SCOPE);
        bindings.put("input",o);
        Reader r = new FileReader("./scripts/"+scriptName);
        Object result = engine.eval(r);
        r.close();
        return result;
    }

}
```

DEXELS

Scripting languages in OSGi

- Use any (?) scripting language Use JSR-223
- Dynamically add, modify and remove scripts Expose it as an OSGi service
- Can declaratively depend on other services
- No external depedencies
- Behave like any other bundle

Generating bundles

- Make up symbolic name
- Make up a version
- Determine Java dependencies

DEXELS

Generate an implementation class

- We need an actual implementation class in the generated bundle.
- Usually it can be pretty simple

Generated java file:

```
package dexels.apachecon;

import dexels.apachecon.api.Evaluator;
import dexels.apachecon.api.base.BaseEvaluator;

public class Script extends BaseEvaluator implements Evaluator {

    public Script(String name) {
        super(name);
    }
}
```

Base class*

```
public abstract class BaseEvaluator implements Evaluator {

    protected final String scriptName;

    public BaseEvaluator(String scriptName) {
        this.scriptName = scriptName;
    }

    @Override
    public Object evaluate(Object o) throws ScriptException, IOException {
        final ScriptEngineManager factory = new ScriptEngineManager();
        final ScriptEngine engine = factory.getEngineByName("js");
        Bindings bindings = engine.getBindings(ScriptContext.ENGINE_SCOPE);
        bindings.put("input",o);
        InputStream is = getClass().getResourceAsStream(scriptName);
        Reader r = new InputStreamReader(is);
        Object result = engine.eval(r);
        r.close();
        return result;
    }

}
```

*The actual implementation is a bit messier

DEXELS

Generate a MANIFEST

```
Manifest-Version: 1.0
Bnd-LastModified: 1352097476937
Bundle-ManifestVersion: 2
Bundle-Name: dexels.apachecon.manualexample
Bundle-SymbolicName: dexels.apachecon.manualexample
Bundle-Version: 1.0.0
Import-Package: dexels.apachecon.api;version="[1.0,2)",dexels.apachecon.
 api.base;version="[1.0,2)"
Private-Package: dexels.apachecon
Tool: Bnd-2.0.0.20121026-120249
```

DEXELS

Scripting languages in OSGi

- Use any (?) scripting language Use JSR-223
- Dynamically add, modify and remove scripts (Generate bundles)
- Expose it as an OSGi service
- Can declaratively depend on other services
- No external depedencies (Generate bundles)

Expose a service to OSGi

- Register manually from an activator
- Use a dependency injection framework

DEXELS

Declarative Services

- Easier to generate XML than Java code
- Tooling (Felix WebConsole)
- Dependency management for free

Generate DS file

```
<?xml version='1.1'?>

<component name="dexels.apachecon.Script">
  <implementation class="dexels.apachecon.Script"/>
  <service>
    <provide interface="dexels.apachecon.api.Evaluator"/>
  </service>
</component>
```

Scripting languages in OSGi

- Use any (?) scripting language Use JSR-223
- Dynamically add, modify and remove scripts (Generate bundles)
- Expose it as an OSGi service (DS)
- Can declaratively depend on other services (DS)
- No external depedencies (Generate bundles)

Finding dependencies

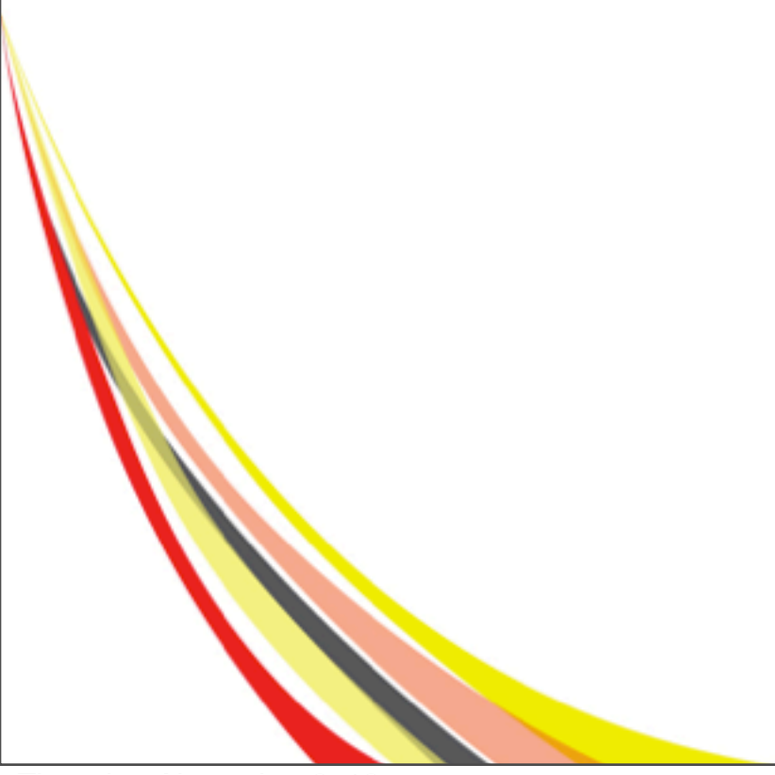
- The scripting language supports this :-) (TSL/Navajo)
- Annotate script :-|
- Use a descriptor file :-(

Declarative Services (DS)

- Determine service dependencies
- Generate a Declarative Service descriptor
- Service will only be activated and exposed when all dependencies are met.
- Generate bind/unbind methods for component class

Compiling Java in OSGi

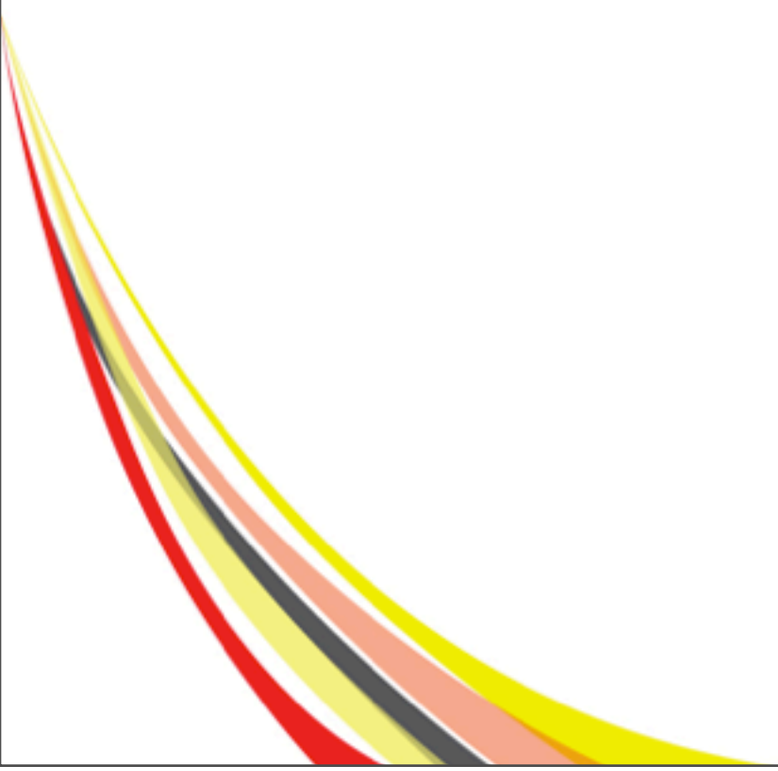
- JSR 199 Compiler API



DEXELS

Compiling Java in OSGi

- What is the classpath?



DEXELS

Compiling Java in OSGi

- Create a custom javax.tools.JavaFileManager
- Use the OSGi Bundle Wiring API to locate exported packages
- Listen to bundle events

DEXELS

Compiling Java in OSGi

- Record all requested packages during compilation
- We've found our package dependencies.. Sort of..
- Only works for scripts that are compiled :-(
- Use Dynamic-ImportPackage if all else fails.

Manage changes

- Manual
- Monitor file system (Felix FileInstall ArtifactTransformer)
- On demand (For example git hooks)

DEMO

- JRuby

Generated Java

```
package script;
import dexels.apachecon.api.Evaluator;
import dexels.apachecon.api.base.BaseEvaluator;

public class Ruby extends BaseEvaluator implements Evaluator {

private dexels.apachecon.billboard.Billboard _billboard;

  public Ruby() {
    super(Ruby.class.getSimpleName(),"ruby",".rb");
  }

     protected String getScriptName() {
        return "script.Ruby";
     }

  public void setbillboard(dexels.apachecon.billboard.Billboard resource) {
    this._billboard = resource;
    setResource("billboard",resource);
  }

  public void clearbillboard(dexels.apachecon.billboard.Billboard resource) {
    this._billboard = null;
    clearResource("billboard",resource);
  }
}
```

DEXELS

Generated DS

```
<?xml version="1.0" encoding="UTF-8"?>
<scr:component immediate="true" name="script.Ruby" xmlns:scr="http://www.osgi.org/xmlns/scr/v1.1.0"
        activate="activate" deactivate="deactivate">
    <implementation class="script.Ruby"/>
    <service>
        <provide interface="dexels.apachecon.api.Evaluator"/>
    </service>
    <property name="name" type="String" value="Ruby"/>
    <reference cardinality="1..1" name="billboard" interface="dexels.apachecon.billboard.Billboard"
        unbind="clearbillboard" policy="static" bind="setbillboard"/>
</scr:component>
```

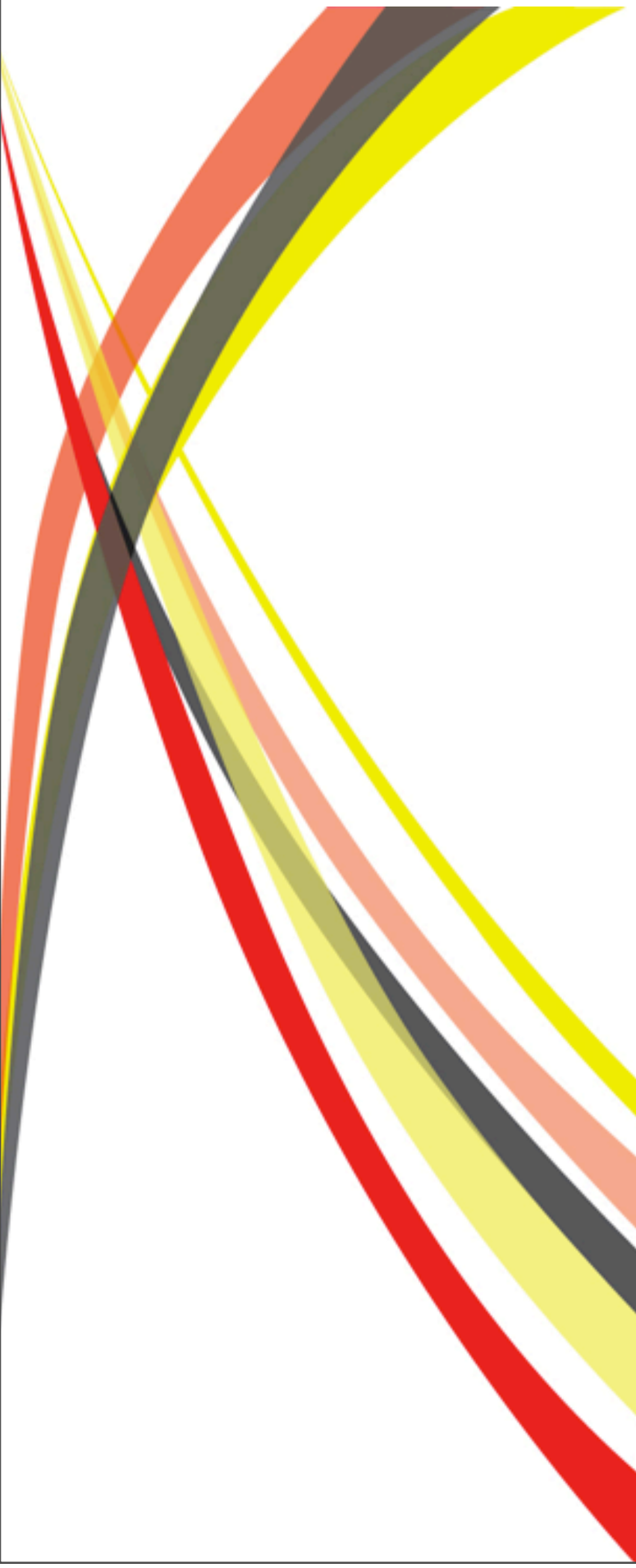
Current limitations

- Try a language with a byte code compiler (like Scala)
- Class.forName / ThreadContext classloaders call still cause trouble
- Determining dependencies (both code depedencies and service dependencies) is really script dependent
- Split packages will probably fail

Conclusions

- Different views on dynamic code
- Creating bundles on the fly *is* possible and not that hard*
- Navajo runs at high volume with about 5000 bundles
- ‘Intelligent’ configuration files

DEXELS



Questions?

Frank Lyaruu

frank@dexels.com
Twitter: @lyaruu

Demo code:
github.com/Dexels/apachecon

Navajo code:
github.com/Dexels/navajo

DEXELS