Hot Deployable Cocoon Blocks with OSGi

Daniel Fagerström
danielf@nada.kth.se

Overview

- Why?
- How?
  - Bundelization
  - Component wiring
  - Runtime environment
- When?

Why?

- OSGi is the standard plugin framework
  - Reuse bundles from other projects
  - Other projects can reuse Cocoon bundles
- Dynamically updateable bundles
  - 24/7
  - Convenient for customer installations
  - Faster development
- Classloader isolation
  - Possible to use multiple versions of the same library

Deja vú?

- Earlier attempts with OSGi
- Focused on the Servlet Service FW
- OSGi more mature for enterprise apps now

Modularization in 2.2

- Packaging
- Spring Beans everywhere
- Servlet Service FW
- Deployment in **global** classloader and **global** Spring factory
- No dynamics

How?

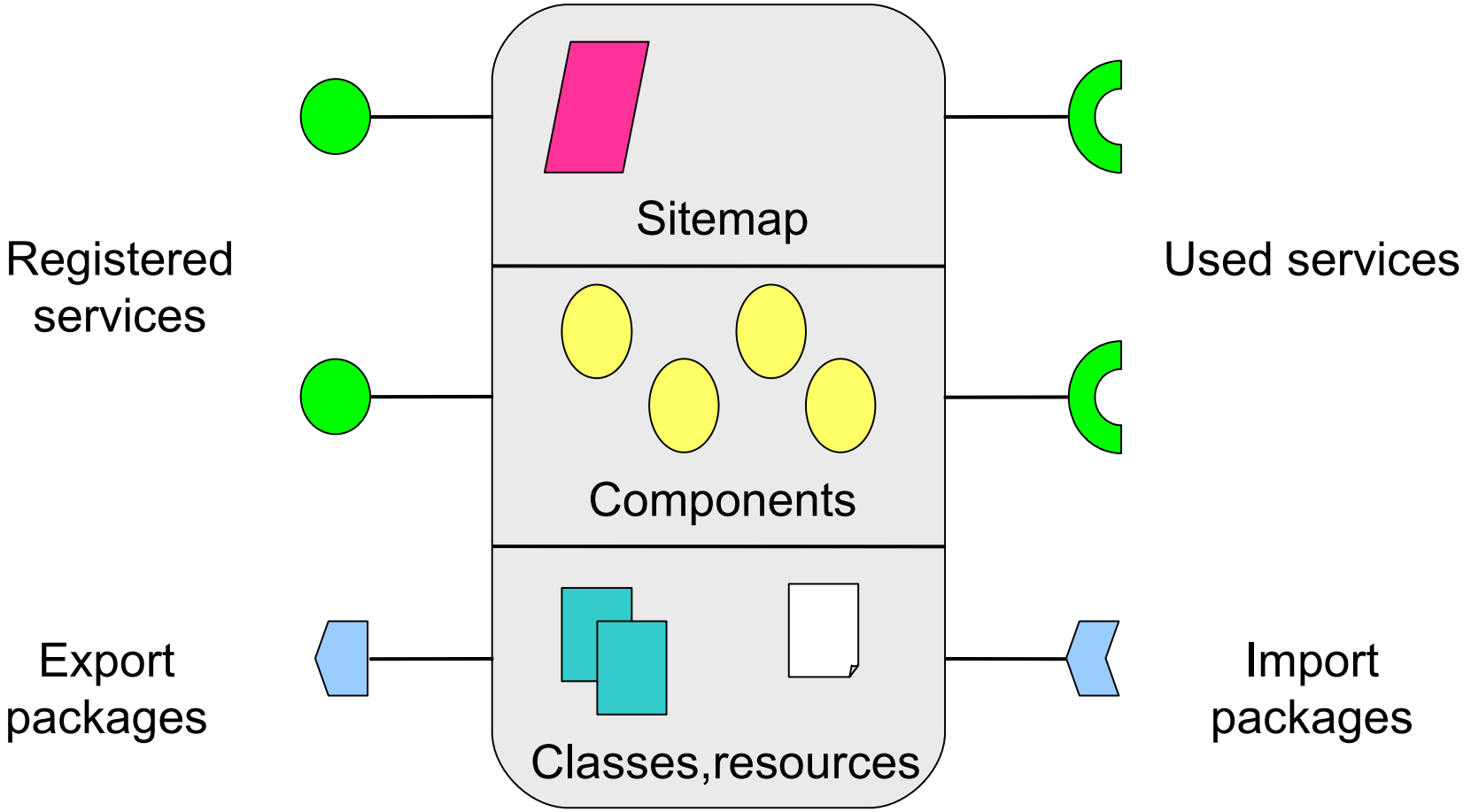
OSGi

- Felix – OSGi Framework
- Maven Bundle - Plugin for packaging
- Spring-OSGi – Component wiring
- Services from Felix, Spring-OSGi, Equinox, OPS4J

What is a block?

- A packaged application (or part) containing:
  - Libraries and resources
  - Components
  - Sitemap functionality
- Configurable at deploy time
- Might depend on other blocks
- Isolated internals (only partly in 2.2)

What is a Block?



Block Structure

```
myBlock/
  META-INF/
    manifest.mf                 # bundle manifest

    cocoon/
      spring/
        components.xml          # spring components

    spring/
      osgi-services.xml         # spring-osgi - exports & imports

  org/
    apache/
      cocoon/
        ...                     # classes

  COB-INF/
    sitemap.xmap
    ...                         # resources
```

Component Wiring

- Using Spring-OSGi
- One web application context per block
- Exporting and importing beans as OSGi services

Spring-OSGi vs. Declarative Services

Spring-OSGi

- \+ Spring
- \- Not released yet

DS

- \+ Standard
- \+ Several Releases
- \- More container dependencies in the code
- \- More work to integrate with Spring

Exporting a Bean

```
<!-- cocoon-xml-impl -->

<bean
  name="org.apache.cocoon.core.xml.SAXParser"
  class="org.apache.cocoon.core.xml.impl.JaxpSAXParser"
  scope="singleton">
  <property
    name="validate"
    value="false"/>
</bean>

<osgi:service
  ref="org.apache.cocoon.core.xml.SAXParser"
  interface="org.apache.cocoon.core.xml.SAXParser"/>
```

Importing a Bean

```
<!-- cocoon-pipeline-components -->

<osgi:reference
  id="org.apache.cocoon.core.xml.SAXParser"
  interface="org.apache.cocoon.core.xml.SAXParser"/>

<bean
  name="org.apache.cocoon.generation.Generator/file"
  class="org.apache.cocoon.generation.FileGenerator"
  scope="prototype">
  <property
    name="parser"
    ref="org.apache.cocoon.core.xml.SAXParser"/>
</bean>
```

Bean Map

- Whiteboard pattern

Bean Map

```
<!-- cocoon-expression-language-impl -->

<osgi:service
  ref="org.apache.cocoon.el.ExpressionCompiler/js"
  interface="org.apache.cocoon.el.ExpressionCompiler"/>

<osgi:service
  ref="org.apache.cocoon.el.ExpressionCompiler/jexl"
  interface="org.apache.cocoon.el.ExpressionCompiler"/>

<osgi:service
  ref="org.apache.cocoon.el.ExpressionCompiler/jxpath"
  interface="org.apache.cocoon.el.ExpressionCompiler"/>

<osgi:service
  ref="org.apache.cocoon.el.ExpressionCompiler/default"
  interface="org.apache.cocoon.el.ExpressionCompiler"/>
```

Bean Map

```
<!-- cocoon-expression-language-impl -->

<bean id="org.apache.cocoon.el.ExpressionFactory"
  class="org.apache.cocoon.el.impl.DefaultExpressionFactory">
  <property name="expressionCompilers">
    <cosgi:map
      interface="org.apache.cocoon.el.ExpressionCompiler"/>
  </property>
</bean>

<osgi:service ref="org.apache.cocoon.el.ExpressionFactory"
  interface="org.apache.cocoon.el.ExpressionFactory"/>
```

Service Events

- Fine grained control of addition and removal of services

Service Events

```
<!-- cocoon-servlet-service-demo1 -->

<osgi:service
  ref="org.apache.cocoon.servletservice.demo1.servlet"
  interface="javax.servlet.Servlet">
  <osgi:service-properties>
    <prop key="mountPath">/test1</prop>
  </osgi:service-properties>
</osgi:service>

<!-- cocoon-servlet-service-demo2 -->

<osgi:service
  ref="org.apache.cocoon.servletservice.demo2.servlet"
  interface="javax.servlet.Servlet">
  <osgi:service-properties>
    <prop key="mountPath">/test2</prop>
  </osgi:service-properties>
</osgi:service>
```

Service Events

```
<!-- cocoon-servlet-service-impl -->

<osgi:reference
  id="httpService"
  interface="org.osgi.service.http.HttpService"/>

<osgi:collection
  id="servletService"
  interface="javax.servlet.Servlet">
  <osgi:listener
    bind-method="setServlet"
    unbind-method="unsetServlet"
    ref="servletListener"/>
</osgi:collection>

<bean
  id="servletListener"
  class="org.apache.cocoon.servletservice.osgi.Activator">
  <property
    name="httpService"
    ref="httpService"/>
</bean>
```

Tunnelling a Prototype

- No correspondence to prototype scope in OSGi
- Export a factory bean instead

Tunnelling a Prototype

```
<!-- cocoon-pipeline-components -->

<bean
  name="org.apache.cocoon.generation.Generator/file"
  class="org.apache.cocoon.generation.FileGenerator"
  scope="prototype">
  <property name="parser"
    ref="org.apache.cocoon.core.xml.SAXParser"/>
</bean>

<cosgi:service
  ref="org.apache.cocoon.generation.Generator/file"
  interface="org.apache.cocoon.generation.Generator"
  factory-export="true"/>
```

Tunnelling a Prototype

```
<!-- cocoon-core-main-sample -->

<cosgi:reference
  name="org.apache.cocoon.generation.Generator/file"
  interface="org.apache.cocoon.generation.Generator"
  factory-export="true"/>
```

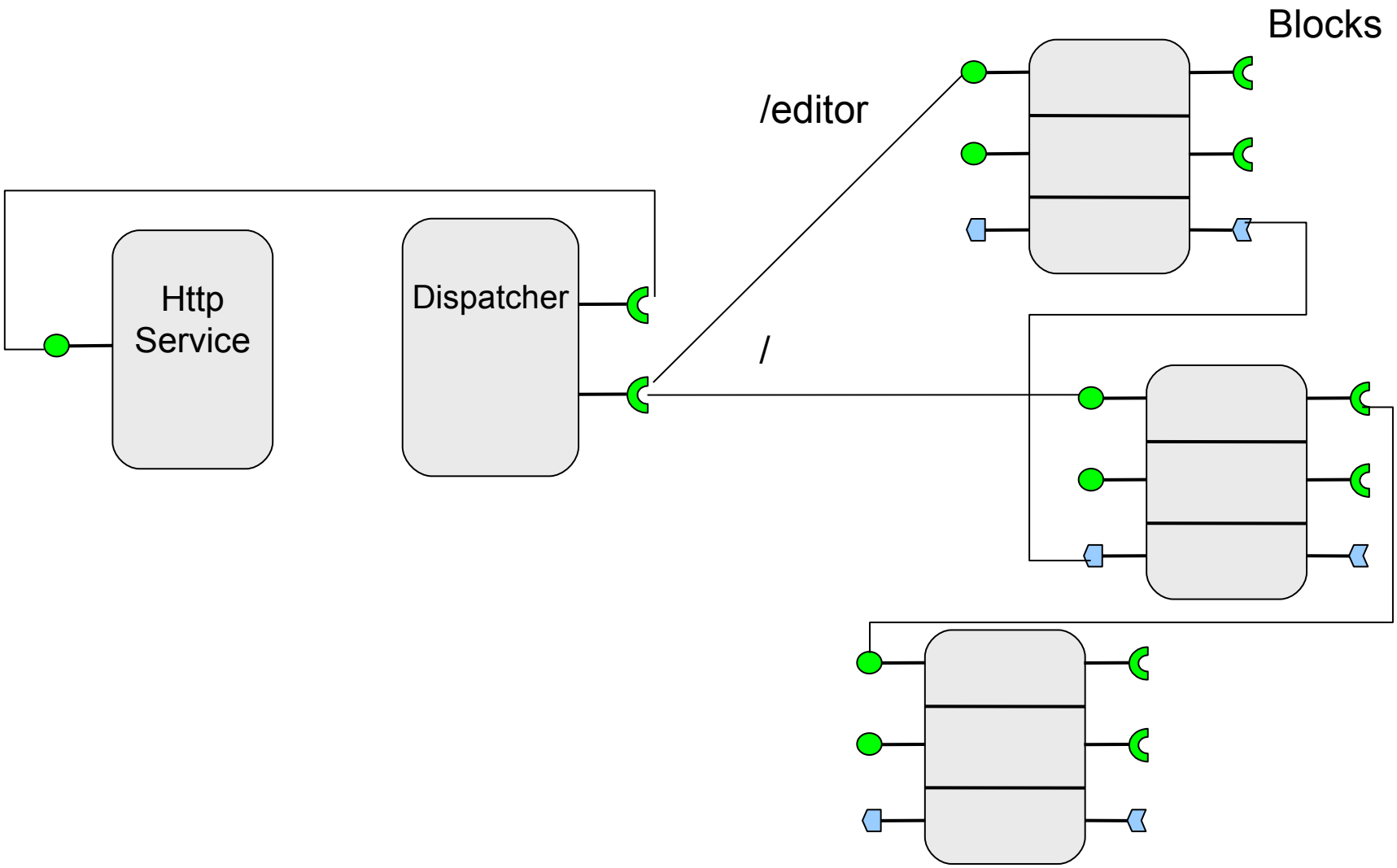
Service Manager?

- A sitemap may need many beans
- Many service imports
- Let the bundle local Spring context fall back to the OSGi service registry?
  - Will that work with declarative wiring?

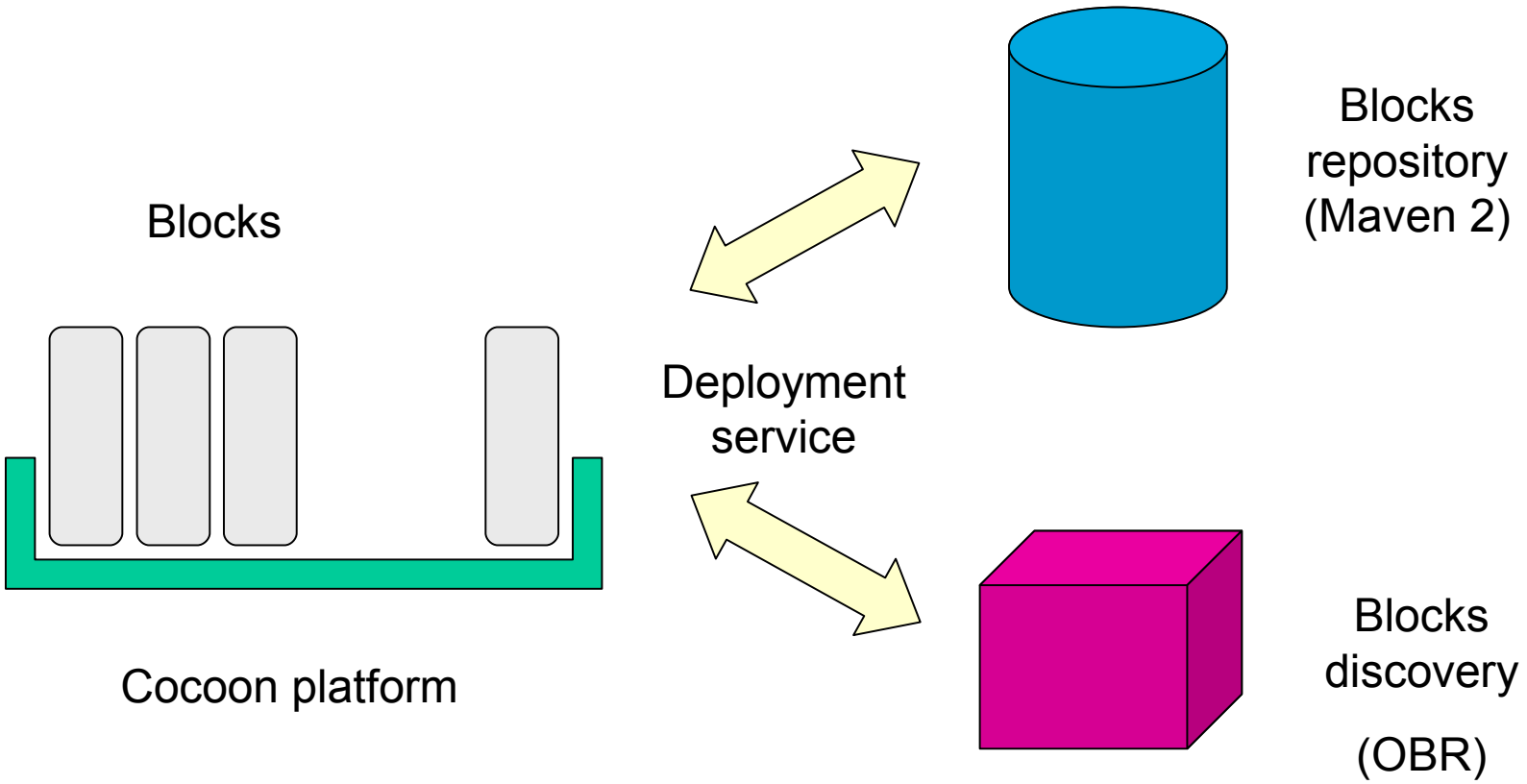
Cocoon-OSGi Architecture

- Http Service
- Servlet container embedding?
- Other services?
  - Config
  - Log
  - ...

Cocoon-OSGi Architecture



Deployment architecture



When?

- One man show this far
- Can live together with ”ordinary” Cocoon
- Spring-OSGi expected to release 1.0.0 in December
- `http://svn.apache.org/repos/asf/cocoon/whiteboard/osgi/`

Conclusion

- Standard plugin
- Classloader isolation
- Hot deployable blocks