Cocoon Blocks

Daniel Fagerström
danielf@nada.kth.se
Same talk as last year?

- Changed architecture 3 times since last time and rewritten the implementation a couple of times
- The latest incarnation is based on the Spring framework and the servlet set of APIs
- What I will describe is now part of Cocoon 2.2
Motivation

• Plugin architecture
• Webapp reuse
• Isolated internals in the blocks
• Simplify using Cocoon together with other Servlet frameworks
Overview

• The big picture

• Architecture
  – Focus on the webapp reuse part
  – Examples

• Current state and next steps
Blocks

• A plugin architecture is needed
• Designed by Stefano and the rest of the community 4+ years ago
• Compile time blocks for a few years, but no external contracts
• Several prototypes the last 1.5 years
• Essentially back compatible, a new integration level: package and reuse applications
What is a block?

• A packaged application (or part) containing:
  – Libraries and resources
  – Components
  – Webapp functionality

• Configurable at deploy time

• Might depend on other blocks

• Isolated internals (only partly in 2.2)
What is a Block?

- Exported components
- Servlets
- Components
- Classes, resources
- Used components
Deployment architecture

Cocoon platform

Blocks

Deployment service

Blocks repository (Maven 2)

Blocks discovery
Block Architecture

- Built upon Spring and Maven
- A block is a Maven module
  - Packaging format
  - Components
  - Servlet(s)
  - Resources
  - Libraries
Block structure

myblock/

META-INF/

  legacy/
    components.xconf # Avalon conf

properties/

  component.properties

spring/

  components.xml # Spring conf (incl block servlet)

COB-INF/

  sitemap.xmap # block sitemap

resources/

  ...

org/apache/cocoon/myblock/ # classes

  foo.class

  ...

Components in blocks

- Exported to and managed in a global Spring container
- Now the component configurations are copied from the blocks to the global Spring configuration by cocoon:deploy
- Reading the configuration from the block would be preferable
Webapps in blocks

• As usual
• Spring managed Servlets
• Adds
  – Call servlets (sitemaps) in connected blocks
  – Use block deploy time attributes
  – Extend blocks (with polymorphism)
Block architecture

Dispatcher Servlet

Configured in web.xml

Spring container

Block Servlets
Based on the Servlet API

- No new API
- The BlockServlet is a Spring managed Servlet that sets up a minimal Servlet container for an embedded Servlet (e.g. SitemapServlet)
- Block properties --> Servlet init params
- Block connections --> named dispatchers
- Can be used with any servlet, nothing Cocoon specific
Wiring

blog

editor:

cmsURL: http://mycms.com/...

myeditor

Uses

myblog

super:

mountPath: /blog/danielf/

Extends
BlockServlet configuration

<beans xmlns="http://www.springframework.org/schema/beans">
    <bean id="org.apache.cocoon.blocks.blog" class="org.apache.cocoon.blocks.BlockServlet">
        <property name="mountPath" value="/test1"/>
        <property name="blockServletClass" value="org.apache.cocoon.sitemap.SitemapServlet"/>
        <property name="properties">
            <map>
                <entry key="cmsURL" value="http://mycms.com/test"/>
            </map>
        </property>
        <property name="connections">
            <map>
                <entry key="editor" value-ref="org.apache.cocoon.blocks.editor"/>
            </map>
        </property>
    </bean>
</beans>
Deployment configuration

# blog.properties

## configure the blog block

org.apache.cocoon.blog.properties.cmsURL=
    http://mycvs.com/danielf

org.apache.cocoon.blog.connections.editor=
    com.mycms.myeditor

## configure my extended version

com.mycms.myblog.mountPath=
    /blog/danielf

com.mycms.myblog.connections.super=
    org.apache.cocoon.blog
Block protocol

block:/foo.xml
  – root sitemap in current block
block:./bar.xml
  – current sitemap in current block (not yet)
block:editor:/foo.xml
  – root sitemap in editor block
block:super:/foo.xml
  – root sitemap in extended block
Block properties, paths

`{block-property: cmsURL}`
- Block property in sitemap (input module)

`{cmsURL}`
- Block property in component configuration

`{block-path: myblog:/start}`

--> `/blog/danielf/start`
- “Absolutizes” block protocol URIs to mounted URIs, used in link transformer
Sitemap polymorphism

blog

myblog

Extends

“skin.xsl” → read “skin.xsl”

Empty

"skin.xsl"
Sitemap polymorphism

blog

myblog

Extends

"skin.xsl" → read "skin.xsl"

Override

"skin.xsl" → read "myskin.xsl"
Sitemap polymorphism

```
blog
Extends
myblog

"start.xml"

"skin.xsl" →
read "skin.xsl"

"*.xml" →
generate "{1}"
transform
"block:/skin.xsl"
serialize

"skin.xsl" →
read "myskin.xsl"
```
Scenario

• Download blog block
• Deploy with parameters (or use default)
  – Test
• Create empty extension (Maven archetype)
  – Test
• Override some default or example rule
  – Test
• …
Summary

Blocks gives us:

• Binary application packages
  – Classes & resources
  – Components
  – Webapp functionality
• Parameterizable applications
• Reusability by extension
• Dependency handling between applications
Current state

• Implementation in Cocoon 2.2
• Stabilize it, use it for the samples
Next steps

• 3.0
  – OSGi based
  – Uses ”official” Spring-OSGI bridge
  – class loader isolation
  – partial hot plugability