



INDOQA

Cocoon 2.2 - A long Journey

Keynote - Cocoon GetTogether Rome 2007 - October 5th

Apache Cocoon Committer & PMC Chair
Member of the Apache Software Foundation
reinhard@apache.org

Reinhard Pötz

Indoqa Software Design und Beratung GmbH
Founder and Managing Director
<http://www.indoqa.com>

Where have we started from?

- Cocoon 2.0
 - SAX-based pipelines
 - sitemap
- Cocoon 2.1
 - Control Flow (Flowscript)
 - Blocks as modularization unit for Cocoon's build system



Pains

- maintain our own component container
- no deployment modularization unit
- no (real) contract between sub-applications (at this time implemented as sitemaps)
- no reusable pipelines
- difficult configuration
- too much XML/XSLT/XMAP/Flowscript development



Container



Our needs ...

- Should we as web application framework really maintain or own component container?
- How to implement virtual sitemap components?
 - Virtual Generator: Generator + Transformer(s)
 - Virtual Transformer: Transformer + Tranformer + ...
 - Virtual Serializer: Transformer(s) + Serializer



... our long way to Spring 2.0

- first ... from ECM to Fortress
- then ... moved on to ECM++
- then ... the Spring-Bridge
- finally ... replace ECM++ with Spring 2.0
 - less (difficult) code to maintain
 - IoC container - Dependency injection
 - reuse the wide range of functionality of Spring (e.g. AOP, persistence framework support, etc.)
 - no difference between Cocoon components and Spring beans
 - no virtual sitemap components



Component configurations

Spring bean definition

```
<bean name="org.apache.cocoon.matching.Matcher/doc-exists"  
      class="com.mycompany.DocumentExistsMatcher">  
</bean>
```

- naming convention:
interface name + / + shortname
- put it into /META-INF/cocoon/spring/*.xml



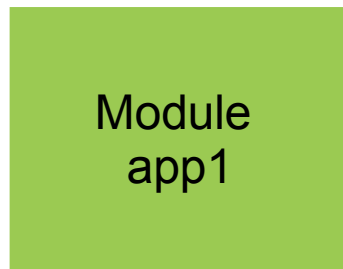
Contracts



Modular web application

Module app1

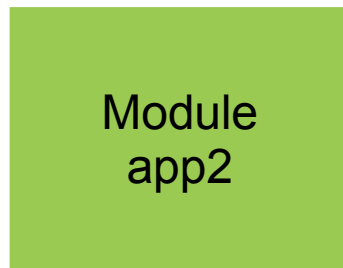
- mounted at /app1
- connection to the module "our-skin"



Module skin
- mounted at /skin

Module app2

- mounted at /app2
- connection to the module "our-skin"



Module
our-skin



The Cocoon 2.1 solution

Root Sitemap

```
<map:match pattern="skin/**">
  <map:mount src="skin/sitemap.xmap" uri-prefix="skin"/>
</map:match>
<map:match pattern="app1/**">
  <map:mount src="app1/sitemap.xmap" uri-prefix="skin"/>
</map:match>
<map:match pattern="app2/**">
  <map:mount src="app2/sitemap.xmap" uri-prefix="skin"/>
</map:match>
```

App1 Sitemap

```
<map:match pattern="bar">
  <map:generate src="bar.xml"/>
  <map:transform src="cocoon://skin/foo.xslt"/>
  <map:serialize/>
</map:match>
```



What's the problem?

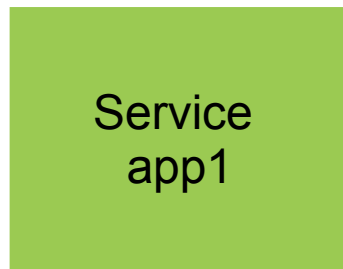
- Using the `cocoon://` protocol to access other sitemaps breaks IoC
- ... since the caller has to know details about where the callee is deployed (i.e. the URI of the skin module)
 - you cannot enforce this **contract**
 - what happens if the “skin” module is moved to another URI
 - how could you reuse the “skin” module (extension)



Use Servlet-Services instead :-)

Servlet-Service app1

- mounted at /app1
- connection to "our-skin" -> "skin"

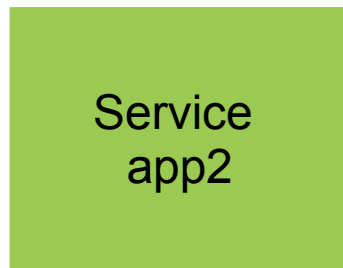


`servlet:skin:/abc`

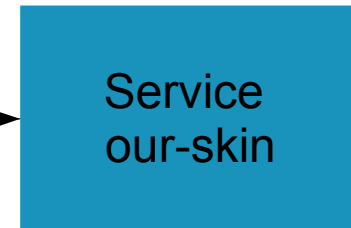
Servlet-Service skin
- mounted at /skin

Servlet-Service app2

- mounted at /app2
- connection to service "our-skin" -> "style"



`servlet:style:/abc`



Move to Servlet-Services

- Use “injected” servlet services – there is no hard dependency on a particular service
- establish contracts between modules
- Use the `servlet:/` protocol which is relative to the servlet for communication
- a Servlet-Service implements `javax.servlet.Servlet`
- implementation of a **Sitemap** Servlet-Service already available
- learn more at http://cocoon.apache.org/2.2/1291_1_1.html



Servlet-Service configuration

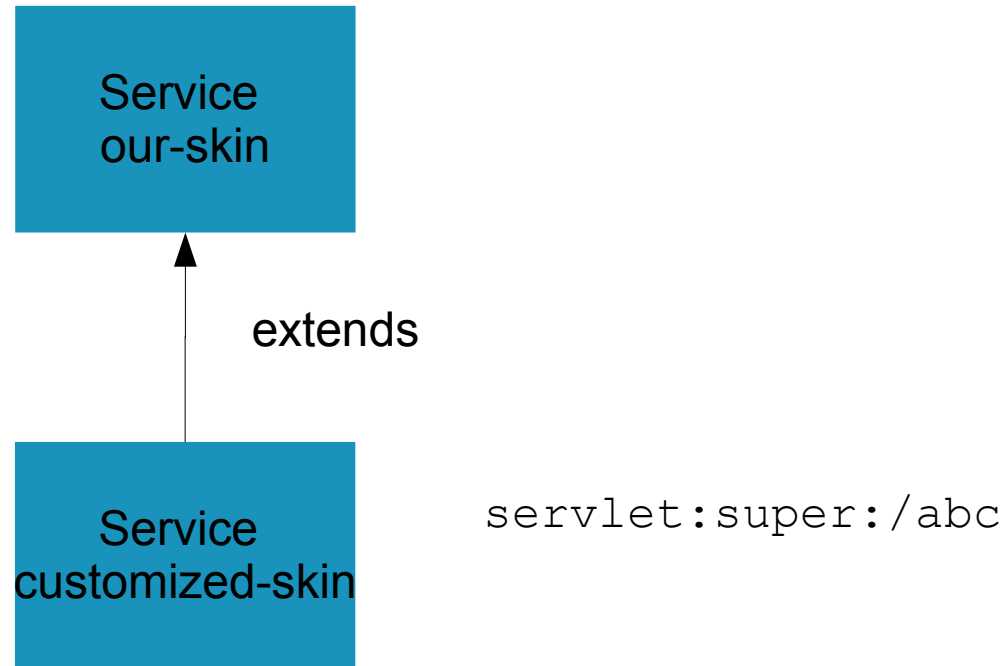
```
<bean id="skin.service" class="o.a.c.sitemap.SitemapServlet">
  <servlet:context mount-path="/skin" context-path="blockcontext:/skin/" />
</bean>
```

```
<bean id="app1.service" class="o.a.c.sitemap.SitemapServlet">
  <servlet:context mount-path="/app1"
    context-path="blockcontext:/app1/">
    <servlet:connections>
      <entry key="skin" value-ref="skin.service" />
    </servlet:connections>
  </servlet:context>
</bean>
```

```
<bean id="app2.service" class="o.a.c.sitemap.SitemapServlet">
  <servlet:context mount-path="/app2"
    context-path="blockcontext:/app2/">
    <servlet:connections>
      <entry key="style" value-ref="skin.service" />
    </servlet:connections>
  </servlet:context>
</bean>
```



Extend relationships



- There is also a fallback mechanism, i.e. if a request can't be served by the 'customized-skin' service, it is delegated to 'our-skin'



Super servlet-service configuration

```
<bean id="skin.service" class="o.a.c.sitemap.SitemapServlet">  
  <servlet:context mount-path="/skin" context-path="blockcontext:/skin/" />  
</bean>
```

```
<bean id="customized-skin.service" class="o.a.c.sitemap.SitemapServlet">  
  <servlet:context mount-path="/customized-skin"  
    context-path="blockcontext:/customized-skin/">  
    <servlet:connections>  
      <entry key="super" value-ref="skin.service" />  
    </servlet:connections>  
  </servlet:context>  
</bean>
```



Reusable pipelines - a use case

A skinning pipeline ...

```
<map:match pattern="skinning-pipeline-to-be-reused">  
  <map:transform src="myXsltTransformation.xslt"/>  
  <map:transform src="strip-namespaces.xslt"/>  
  <map:serialize type="xhtml"/>  
</map:match>
```

- **use** `<map:resource>` + `<map:call resource="">` **but ...**
 - it is only available within the sitemap where it is defined
 - context is shared



Reuseable pipelines

- (re)use pipeline fragments
 - Generator + Transformer(s)
 - Transformer + Transformer (+ Transformer + ...)
 - Transformer(s) + Serializer
- replaces the concept of virtual sitemap components
- implemented based on the servlet-service concept
- special sitemap components that use servlet-services to access other pipelines
- there is no context sharing (= a fresh request)



Reuseable pipelines - example

A skinning pipeline ... [the callee]

```
<map:match pattern="skinning-pipeline-to-be-reused.service">
  <map:generate src="service-consumer:" />
  <map:transform src="myXsltTransformation.xslt" />
  <map:transform src="strip-namespaces.xslt" />
  <map:serialize />
</map:match>
```

Using the skin ... [the caller]

```
<map:match pattern="callingTransformationService">
  <map:generate src="demo/welcome.xml" />
  <map:transform type="servletService">
    <map:parameter name="service"
      value="servlet:myBlock2:/skinning-pipeline-to-be-reused.service" />
  </map:transform>
  <map:serialize type="xml" />
</map:match>
```



Reuseable pipelines

- warning: implementation hasn't been completed yet
 - no caching (yet)
 - context sharing needs to be defined
 - serialization and deserialization of SAX-events --> unnecessary overhead
- learn more at http://cocoon.apache.org/2.2/1291_1_1.html



Deployment & Configuration

Pains

- no deployment modularization unit - we release a source version of Cocoon only!
- difficult to upgrade
- no standard for deployment
- difficult configuration
- patching cocoon.xconf and root sitemaps



Improved deployment

- from a build modularization unit to a binary modularization unit for deployment
- clear distinction between what is Cocoon's and what's your **own** code
- block deployment means putting it into `/WEB-INF/lib`

- don't confuse blocks with servlet-services - It's not the same!!!



What's a block in 2.2?

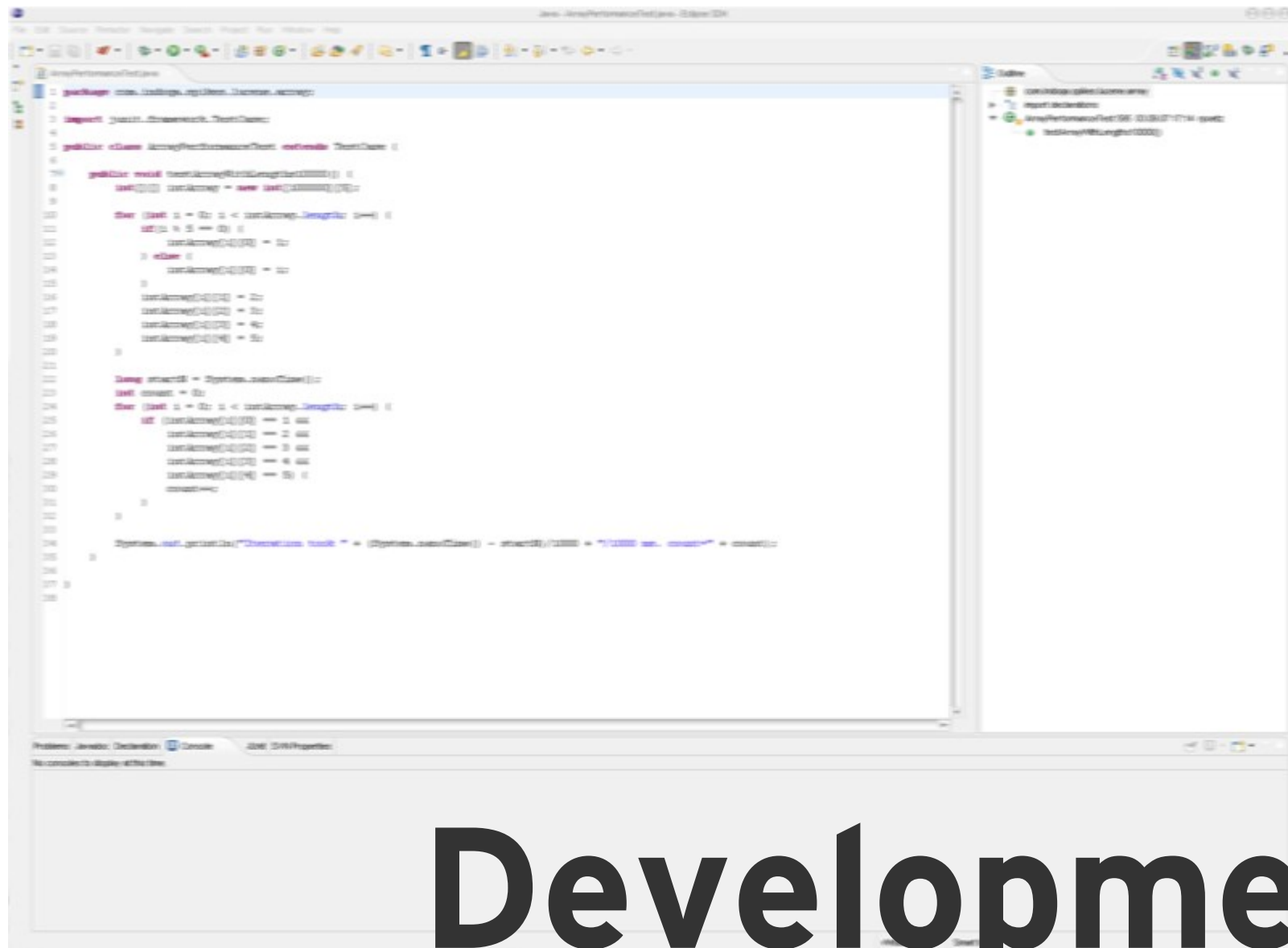
- Blocks are binaries - normal Java Archives (.jar)
- Manifest property: "Cocoon-Block"
- Follows a particular directory structure
(see http://cocoon.apache.org/2.2/core-modules/core/2.2/1263_1_1.html)
- Block can contain Resources, Java classes and component configurations (Spring and/or Avalon)
- Contains a POM where all dependencies on other blocks and libraries are declared
(Note: that's not a hard dependency on Maven because other tools like Ivy or the Maven Ant tasks offer support for them too)



Improved configuration

- easy to provide own configurations, just put
 - Avalon configurations into /META-INF/cocoon/avalon
 - Spring bean definitions into /META-INF/cocoon/spring
- property placeholder configuration
- property configuration overrider
- different configuration levels and running modes
- easy to provide configuration at deployment time
- **see** http://cocoon.apache.org/subprojects/configuration/1.0/spring-configurator/1.0/1304_1_1.html





Development

The problem scope

- People love to solve all problems with
 - XML
 - XSLT
 - Flowscript
 - pipelines that cause side-effects, e.g. the mail-transformer
- How to run a block at development time - building + redeployment isn't really an option
- difficult to start a new project



Cocoon Maven 2 Archetypes

- three different archetypes
 - block
 - plain block
 - webapp
- requires Maven 2.0.6 or above
- for their usage see <http://cocoon.apache.org/maven-plugins/>

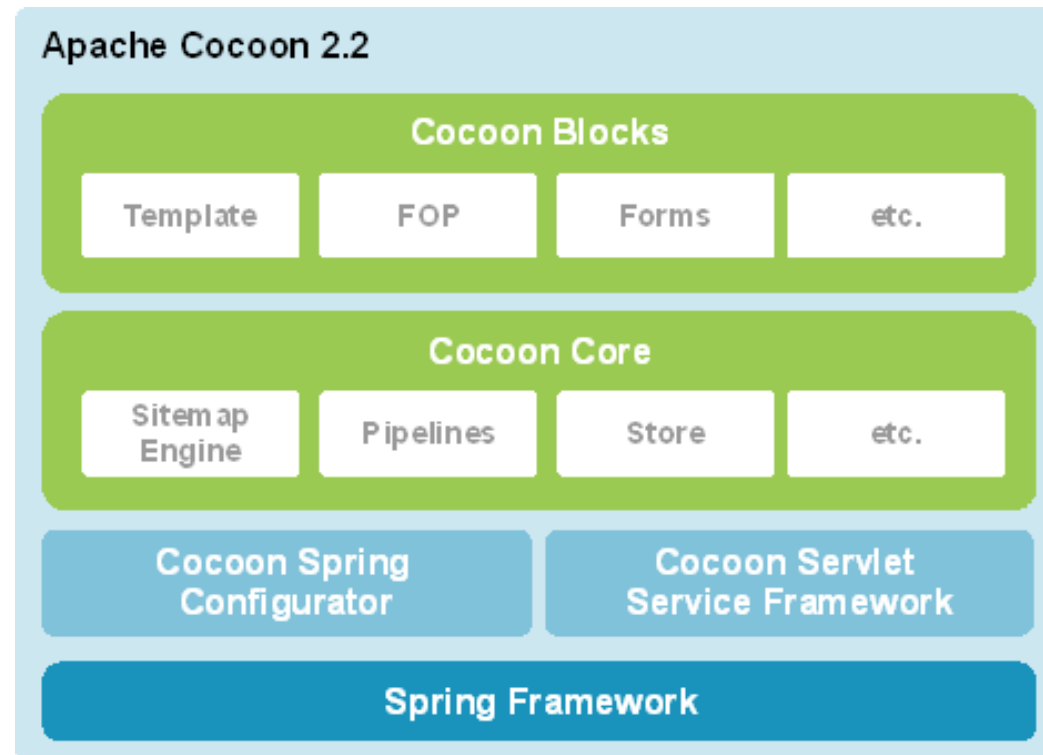


Cocoon Maven 2 Plugin

- run a block as a web application
- automatically detects changes in
 - all resources
 - Java classes
 - Spring application contexts
- possible to mount other blocks as well
- find more about it at <http://cocoon.apache.org/2.2/maven-plugins/maven-plugin/1.0/>



Cocoon 2.2



... checkout our new website at <http://cocoon.apache.org>



Credits

Many thanks to all those who have provided the images and allowed their commercial usage.

- Container: http://www.flickr.com/photo_zoom.gne?id=104795439&size=o
- Shaking hands: <http://www.flickr.com/photos/gaetanlee/159591865/>

