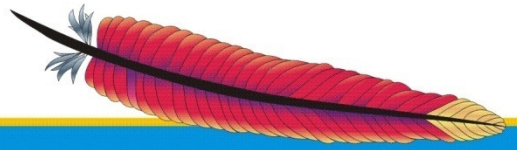
ApacheCon

From A(valon) to O(SGi)

The Future of Modular (Web)Applications

Carsten Ziegeler
cziegeler@apache.org
Day Software

Felix Meschberger
fmeschbe@apache.org
Day Software



Leading the Wave of Open Source

About Felix Meschberger

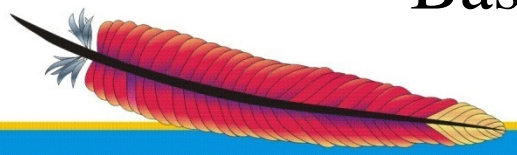
- Committer in some Apache Projects
  - Jackrabbit, Felix, Sling
  - PMC: Felix, Jackrabbit
- Core Developer at Day Software

About Carsten Ziegeler

- Apache Software Foundation Member
  - Cocoon, Excalibur, Pluto, Felix, Incubator, Sling, Sanselan
  - PMC: Cocoon, Incubator, Portals, Felix, Excalibur (Chair)
- Senior Developer at Day Software
- Article/Book Author, Technical Reviewer
- JSR 286 spec group (Portlet API 2.0)

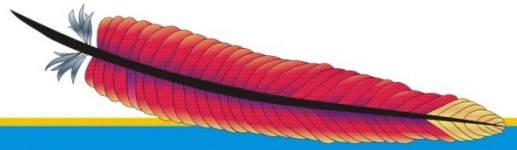
Foreword

- A plethora of “component”/”service” frameworks
  - Making the right decision...
- Focus on the components – not the framework
- We’ll go from A to Ω (Avalon to OSGi)
  - Basic concepts over details



Agenda

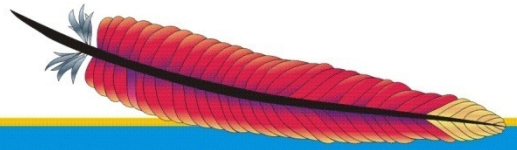
- Background: General Concepts of COP
- Past: Apache Avalon
- Present: OSGi and Apache Felix
- Future: Spring and Apache Sling
- Summary



ApacheCon

Managing Large Systems

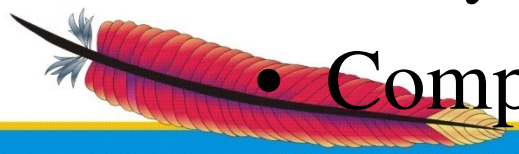
GENERAL CONCEPTS OF COP



Leading the Wave
of Open Source

Managing Complexity

- Modularization
  - Improved quality / robustness
  - Team work
  - Easier problem location
  - Aids deployment and maintenance
  - Extensibility
  - Dynamic systems
- Component oriented programming

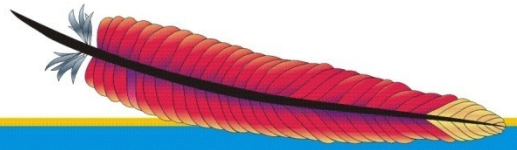


A Component

- Consists of two parts
  - Service/Role (offered functionality) – (Java)Interface
  - (Java)Implementation
- Client knows only about service/role
  - Behaviour
- Separation of concerns
- Managed by a container

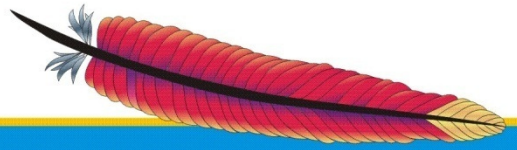
Component Container

- Manages components
  - Central repository
  - Central configuration
  - Connects services with implementations
    - Usually through dependency injection

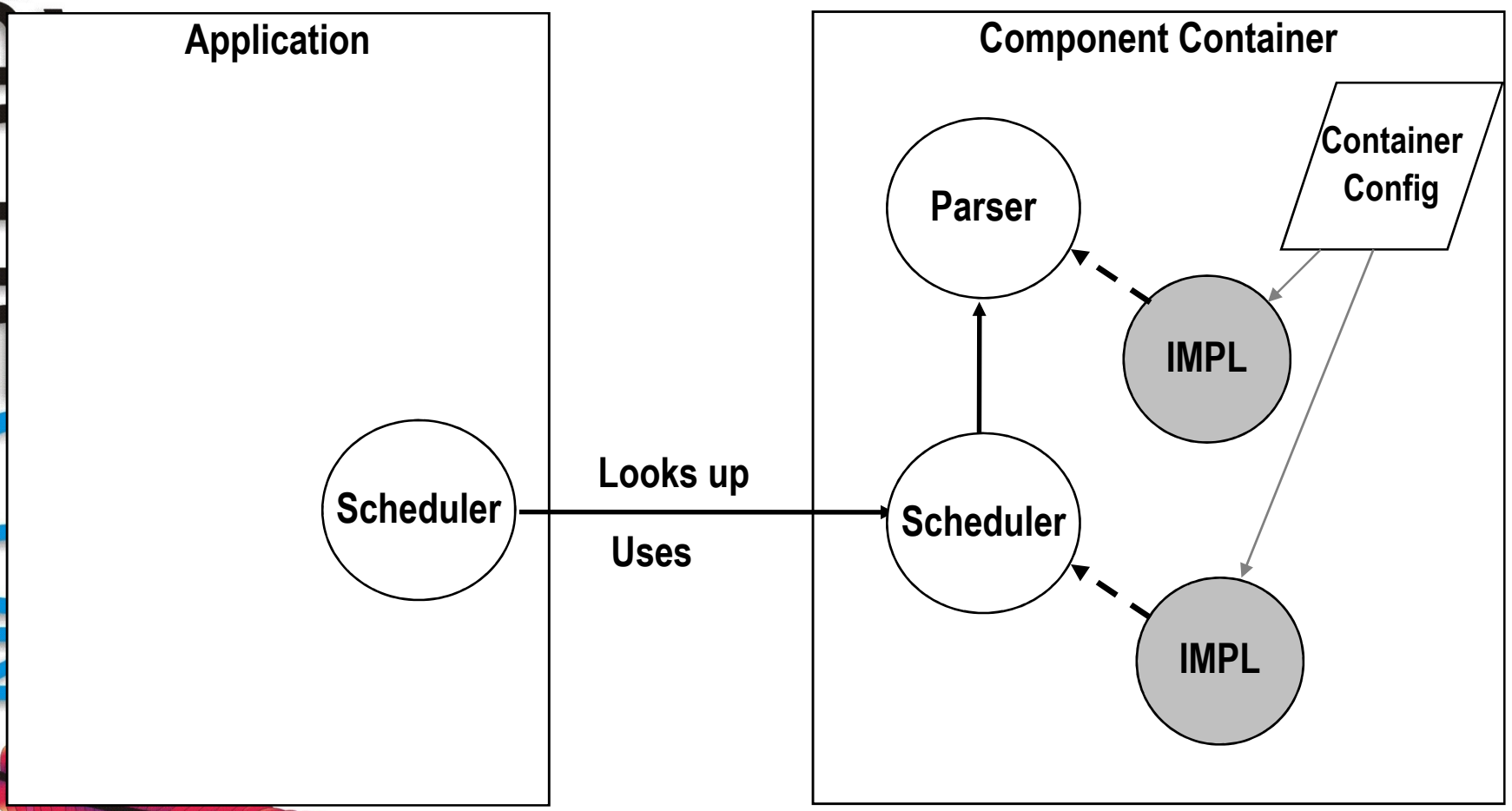


Dependency Injection

- Favours loose coupling
- Makes implementation replacement easy
  - Even at runtime
- Breaks the dreaded “everything depends on everything” problem

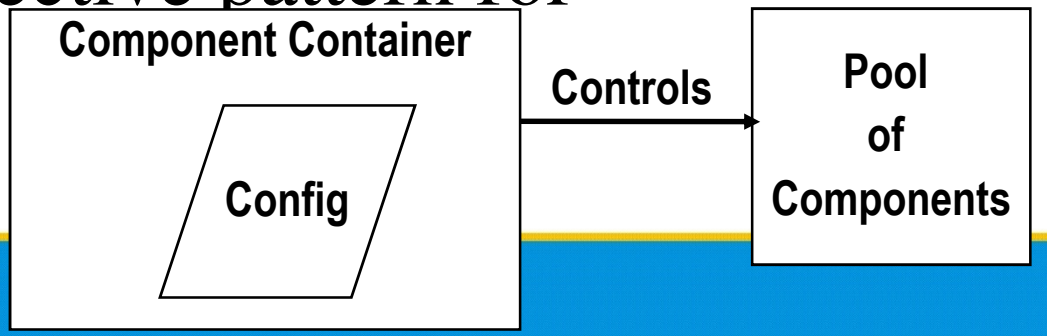


Application Architecture



IoC = Inversion of Control

- Everything is managed by the container
  - Creation of components
  - Configuration/Initialization of a component
  - Component lifecycle
  - Destruction of components
- Simple but effective pattern for development



IoC – Setter Injection

- Required information is passed using setter methods
  - Configuration
  - Other Components

All setters have to be called before the component can be used!

Sample: Scheduler

```
public class SchedulerImpl implements Scheduler {

    public void setParser( SAXParser parser ) { ... }
    public void setConfigFile (String path) { ... }
}
```

IoC – Constructor Injection

- Required information is passed in constructor
  - Configuration
  - Other Components

Constructors can get complex and ambiguous!

Sample: Scheduler ("Good Citizens")

```
public class SchedulerImpl implements Scheduler {

    public SchedulerImpl ( SAXParser parser, String path ) { ... }
}
```

IoC – Interface Injection

- Information is passed using special interfaces

Sample: Scheduler

```
public class SchedulerImpl
        implements Scheduler, Serviceable,  Configurable{

    public void service( ServiceManager manager ) {
        LOOKUP PARSER
    }

    public void configure( Configuration conf ) {
        GET FILENAME
    }
}
```

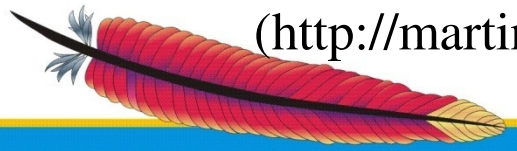
Dependency to the Container Framework!

Inversion of Control – Use It!

"The choice between Service Locator [Interface Injection]and Dependency Injection is less important than the principle of separating service configuration from the use of services within an application.“

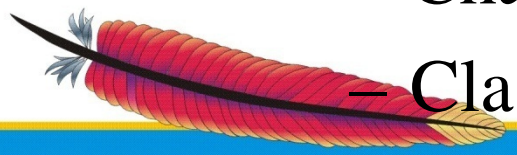
Martin Fowler

(http://martinfowler.com/articles/injection.html)



Flexibility and Modularity

- A good container should
  - be as invisible as possible
  - not impose restrictions on the components
  - offer an API for registering components
- Modularity
  - Updates of (sets of) components
  - Changes during runtime
  - Classloader management

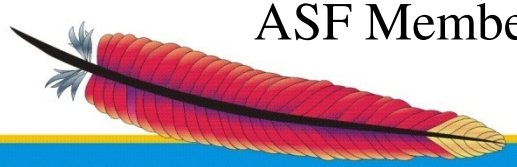


Motivation

"If you ever worked with Avalon,
you know the feeling:
at first it doesn't make any sense
at all. It's a mess of stupid and
very abstract interfaces...but
after a while, a pattern emerges
and it sticks."

Stefano Mazzocchi

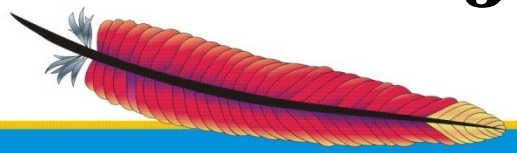
ASF Member



ApacheCon

The Past

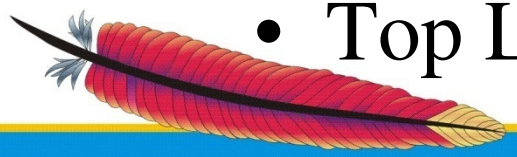
THE APACHE AVALON PROJECT



Leading the Wave
of Open Source

History of Avalon

- Creation of a Java **server** framework
- Apache Initiative
- Reuse of code and components from the various Java Apache projects (1999)
- Later renamed to Avalon (at Jakarta)
- Renamed again to Excalibur ☺
- Top Level Project (excalibur.apache.org)



Original Goals

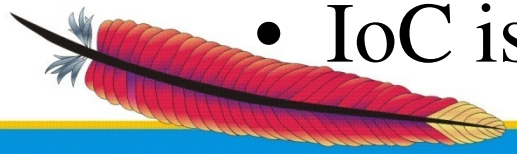
- Java based framework
  - Interfaces, abstract classes, shared modules, patterns
  - Reusable components
  - Shared code
- Web Server Development !
  - Dynamic composition
  - Optimized for multi-threaded environments

Features

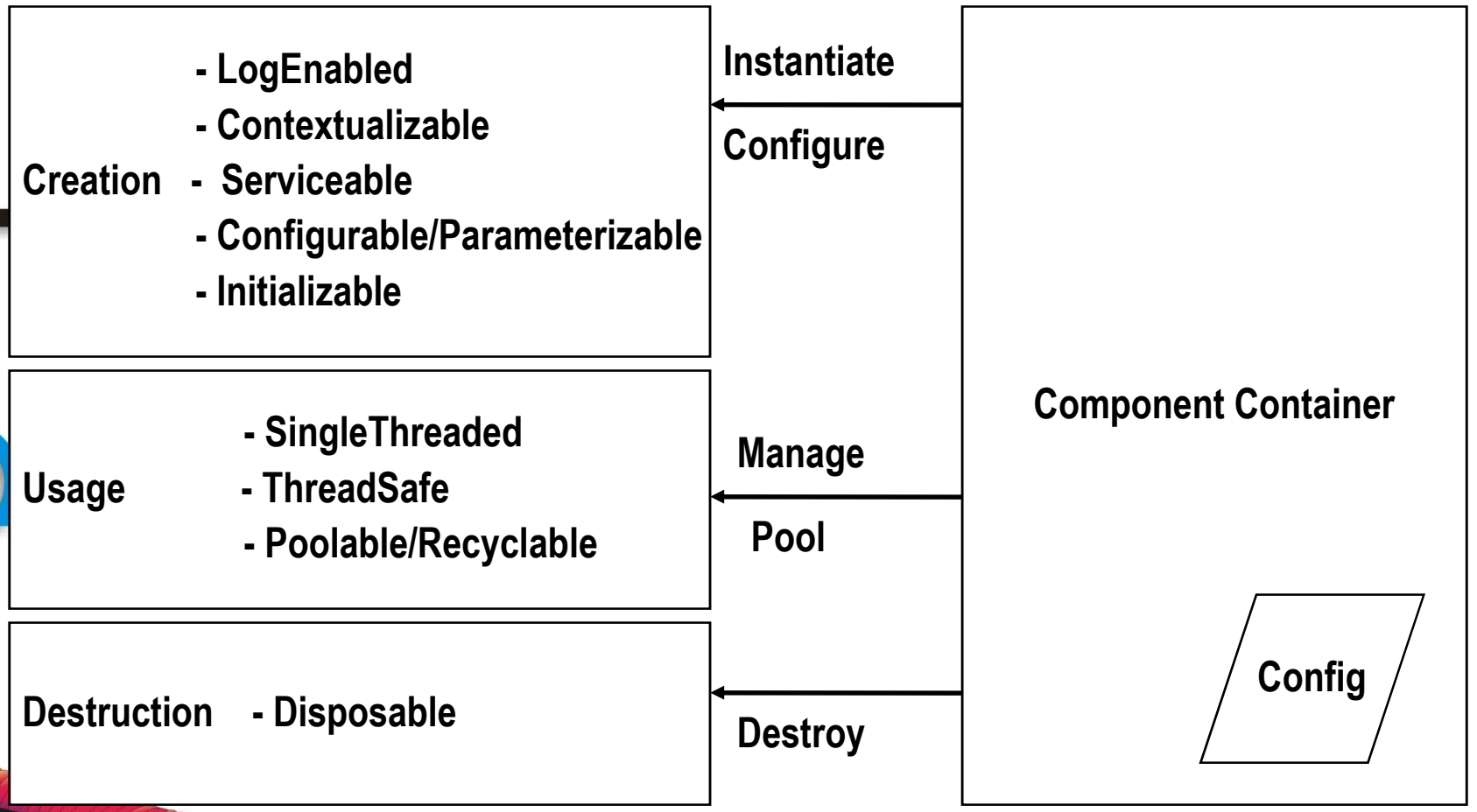
- Separating the application into common Components
  - Accessible using interfaces
  - Different implementations
- Component lifecycle
- Central configuration
- Can be easily integrated with any J2EE framework

IoC in Avalon

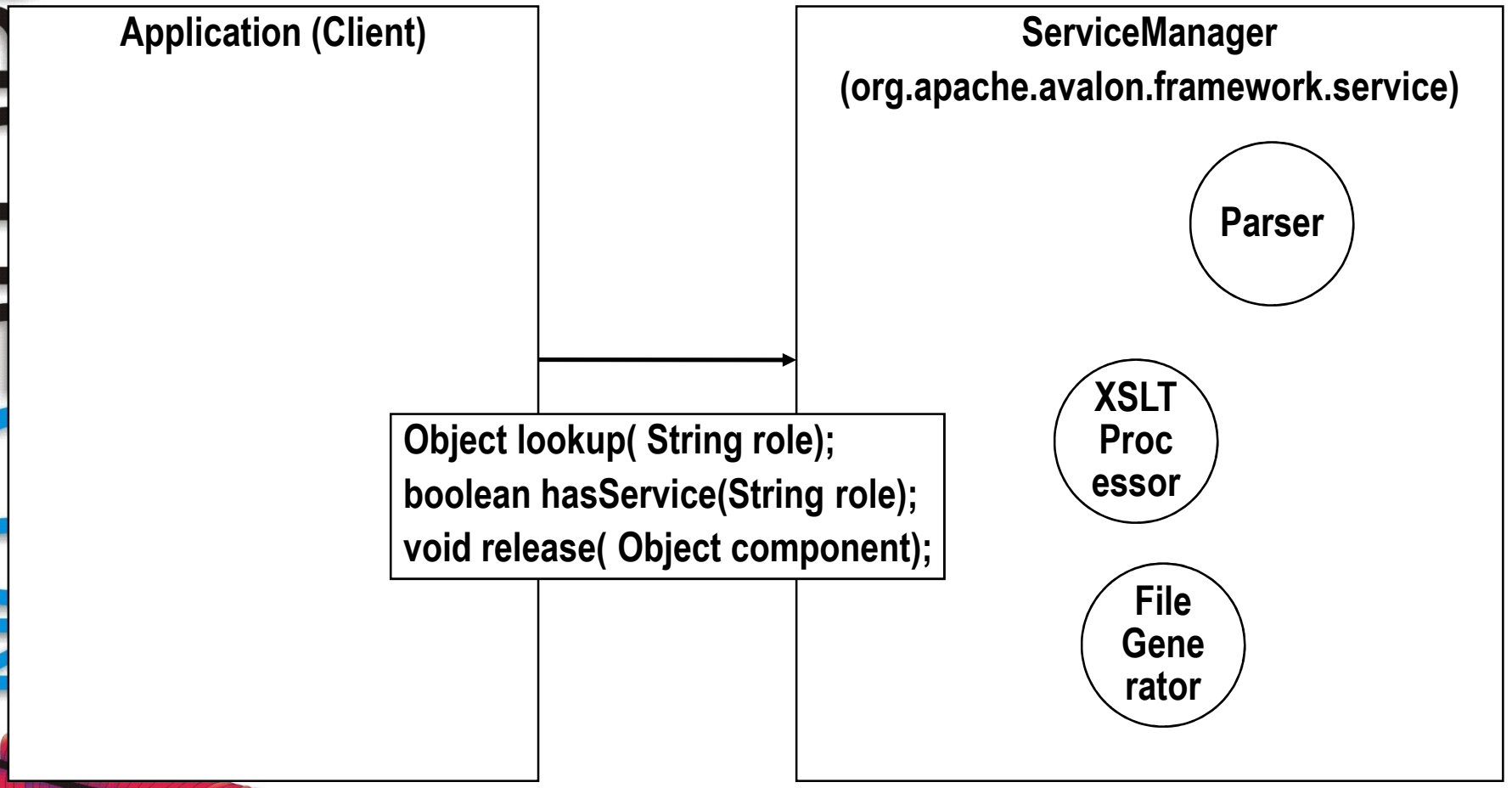
- Interface injection for several concerns
- Service locator
- Constructor injection can be used
  - For some aspects
- Setter injection can be used (config)
- Dynamic coupling of components
- IoC is combined with SoC ☺



Component Lifecycle



Service Manager/Container



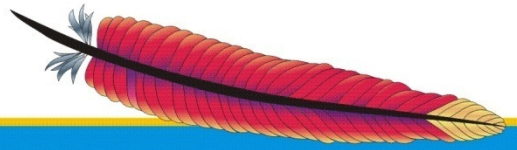
Component/Role Definition

Parser Role

```
package org.apache.excalibur.xml.sax;

import org.xml.sax.*;

public interface SAXParser
{
    String ROLE = SAXParser.class.getName();

    void parse( InputSource in, ContentHandler consumer )
    throws SAXException;

}
```

Using a Component

- Importing the role Interface
- Looking up the role from the service manager
- Using the component
- Releasing the component



Using a Component

Pattern for using a Service Manager

```
import org.apache.excalibur.xml.sax.SAXParser;

import org.xml.sax.*;

public void parse(InputSource document)
{
    SAXParser parser = (SAXParser) this.serviceManager.lookup(SAXParser.ROLE );
    try {

        parser.parse( document, this );

    } catch (Exception ignore) {
    } finally {

        this.serviceManager.release( parser );
    }
}
```

Different Weights of a Role

- Role = Component
  - Parser, XSLT Processor, Session Manager
- Role = Set of components with common behaviour
  - Store (Memory, Disk, MRU), Media Handler (gif, jpeg)
- Distinguished by the lookup role (role vs. role/key)

Configuration (Roles)

```
<role-list>

    <role name="org.apache.excalibur.xml.sax.SAXParser"
          shorthand="xml-parser"
          default-class="org.apache.excalibur.xml.impl.JaxpParser"/>

</role-list>
```

Role (Interface)

Alias

Implementation

Configuration (Components)

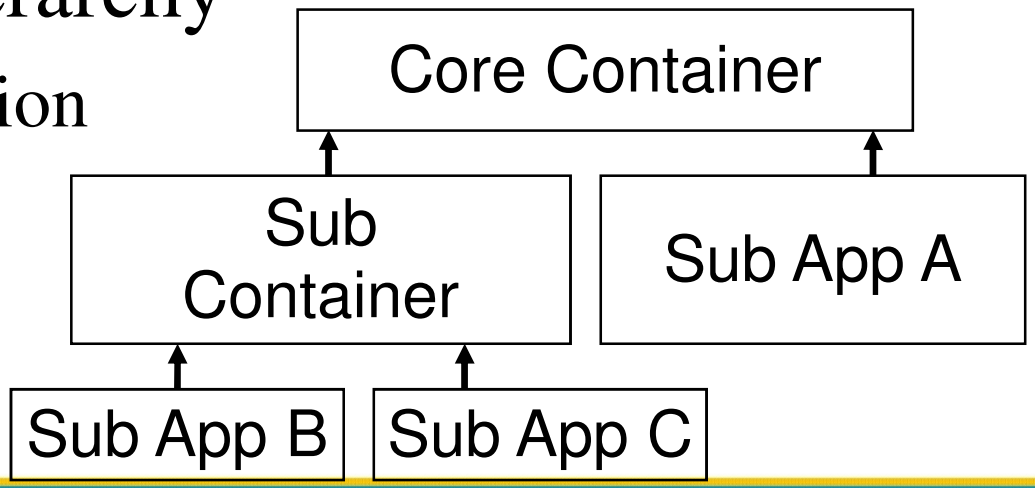
```
<components>
   <xml-parser>

      <parameter name="validate" value="false"/>
      <parameter name="namespace-prefixes" value="false"/>
      <parameter name="stop-on-warning" value="true"/>
      <parameter name="stop-on-recoverable-error" value="true"/>
      <parameter name="reuse-parsers" value="false"/>

   </xml-parser>

</components>
```

Hard-coded alias (in roles file)

xml-parser <-> org.apache.excalibur.xml.Parser

Avalon – The Server Framework

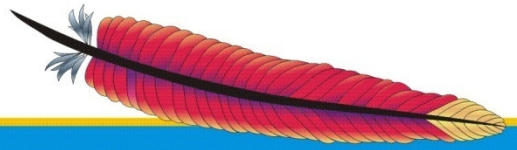
- Dynamic component lookup/assembly
  - Often needed for dynamic request/response based systems
- Container hierarchy
  - Modularization
  - Security



ApacheCon

The Present

OSGI AND APACHE FELIX



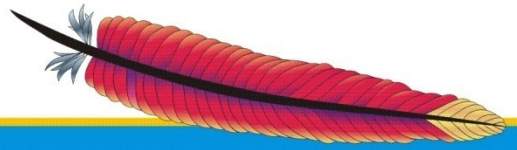
Leading the Wave
of Open Source

OSGi Alliance

- Formerly known as the Open Services Gateway Initiative
- Specification of a framework
  - Dynamic services
  - Simple component model
  - Component lifecycle management
  - Service registration
  - Uses the concept of bundles

An OSGi Bundle I

- Leverages the Java packaging mechanism: JAR file
- Contains Java classes and resources
- Additional meta-data
  - dependencies to other bundles
  - package imports/exports

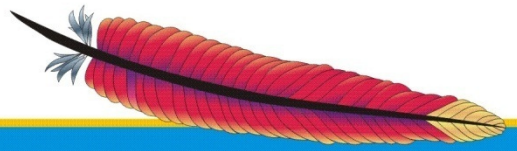


An OSGi Bundle II

- Bundle Activator concept
  - Custom object notified on bundle startup
- Can register services
  - and use other services
- Automatical wiring of bundles
- Solves many modularity problems of todays (web)apps

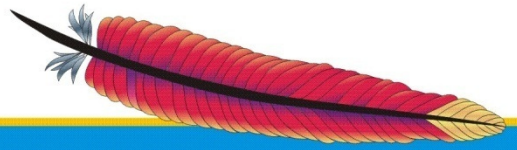
Modularity Requirements I

- A bundle contains more than public classes/API
  - Well defined boundaries (packages)
- A bundle depends on other classes/frameworks etc.
  - Well defined dependencies (packages)



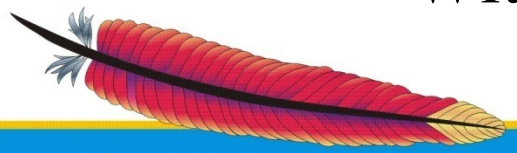
Modularity Requirements II

- A bundle has a version
  - OSGi supports versioning and multi-versions
- Classpath for a bundle is generated by OSGi based on the above information



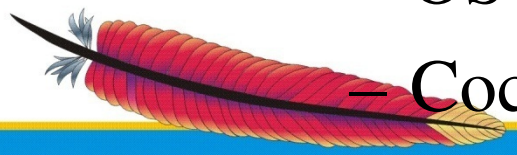
Services

- OSGi offers an API to register services
  - Service is registered by its interface name
  - Implementation is bundle private
  - Several components for same service possible
- Bundles can query services
  - By interface name
  - With additional filters



Configuration Styles

- Jar contains "just" code
  - Additional configuration required
  - Avalon, Spring
  - Cocoon 2.1.x
- Jar contains code **and** configuration
  - Automatic service registration
  - OSGi, Spring + Cocoon Spring Configurator
  - Cocoon 2.2

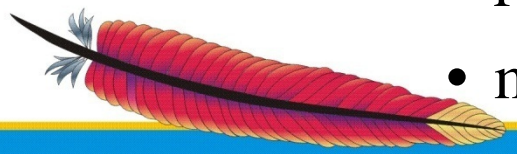


The OSGi Core

- Minimal but sufficient API for services
  - Minimal overhead: Good for simple bundles
  - Requires sometimes a lot of Java coding
  - No support for component management
  - No support for configuration management
- Additional (optional) extensions
  - Declarative Service Specification
  - Configuration Admin Service Specification

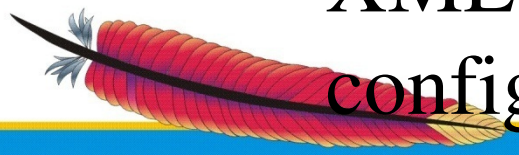
Declarative Service Specification

- XML format for services
  - Services, implementation and references
- Automatic registration on bundle startup
  - Deregistration on bundle stop
- Usage is very straightforward
  - Implementation
    - requires set/unset methods for references
    - might contain special (de)activation methods



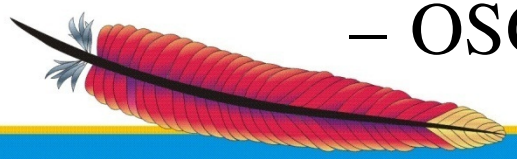
Configuration Admin Service Spec

- Central service for
  - storing and delivering service configurations
  - persistent storage
- API for querying and changing configurations
  - services are updated
- XML meta data description for component configuration



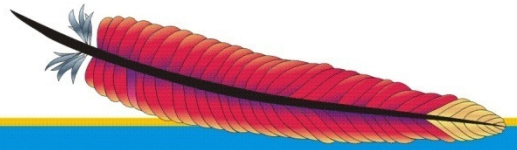
Apache Felix

- Open source implementation of OSGi R4
  - Framework (Core)
  - Services (Compendium)
    - Package Admin, Start Level, Configuration Admin, Declarative Services, Event Admin, Preferences
  - Maven Plugins
  - Shell and other config tools
  - OSGi Bundle Repository (OBR)



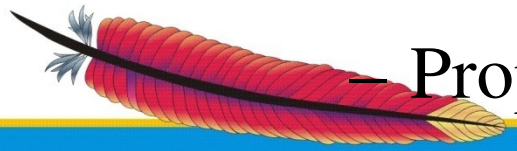
The Maven Bundle Plugin

- Creates a JAR which can be used as a bundle
- Additional meta data
  - is calculated (as far as possible)
  - can be specified in the pom
- Integrates nicely and seamlessly



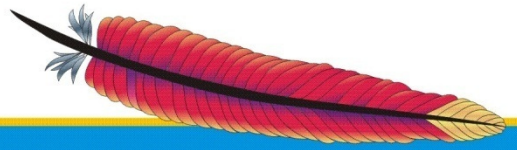
The Maven SCR Plugin

- Generates descriptor files based on annotations
  - Component, service
  - References
  - Class enhancements for simpler usage
- Additional support for the configuration admin
  - Properties



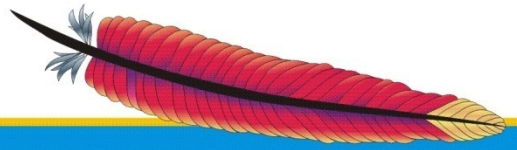
Developing with Apache Felix

- Maven 2
- Maven Bundle Plugin
- Maven SCR Plugin
- (Maven OBR Plugin)



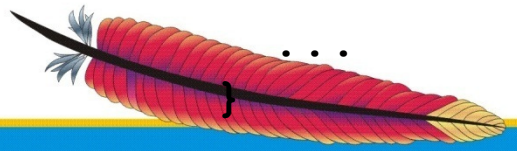
Example Service

- Registering a servlet in a running OSGi environment
- Using provided services
  - LogService for logging
  - HttpService for registering servlets



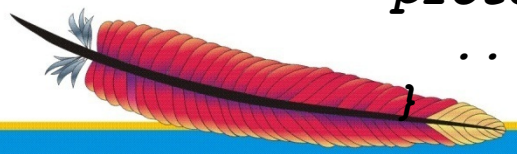
Servlet Service I

```
public class SimpleSlingServlet extends HttpServlet {

  private LogService log;
  private HttpService httpService;

  protected void doGet(....) {
    // nothing Sling/OSGi specific in this method

    // 1. Log
    log.log(LogService.LOG_DEBUG,
    "Processing request, path info=" + req.getPathInfo());

    // 2. Create response
    ...
  }

  ...
}
```



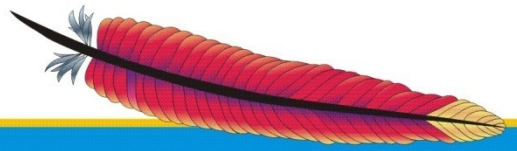
Servlet Service II

```
/**
 * @scr.component
 */
public class SimpleSlingServlet extends HttpServlet {

  /** @scr.reference */
  private LogService log;

  /** @scr.reference */
  private HttpService httpService;

  protected void bindLog(LogService l) {
    ..
  }

  protected void unbindLog(LogService l) {
    ..
  }
```



Servlet Service III

```
public class SimpleSlingServlet extends HttpServlet {

  protected void activate(ComponentContext ctx) {
    httpService.registerServlet("/test", this, null, null);
  }

  protected void deactivate(ComponentContext ctx) {
    httpService.unregister("/test“);
  }
}
```



Servlet Service IV

```
/**
 * @scr.property name="path" value="/test"
 */
public class SimpleSlingServlet extends HttpServlet {

  protected void activate(ComponentContext ctx) {
    String myPath = (String)ctx.getProperties().
                                    get("path");
  }
}
```

Maven Plugin Usage I

```
<plugin>
  <groupId>org.apache.felix</groupId>
  <artifactId>maven-scr-plugin</artifactId>
  <executions>
      <execution>
          <id>generate-scr-scrdescriptor</id>
          <goals><goal>scr</goal></goals>
      </execution>
  </executions>
</plugin>
```

Maven Plugin Usage II

```
<plugin>
  <groupId>org.apache.felix</groupId>
  <artifactId>maven-bundle-plugin</artifactId>
  <extensions>true</extensions>
  <configuration>
    <instructions>
      <Private-Package>
        com.day.sling.examples.*
      </Private-Package>
    </instructions>
  </configuration>
</plugin>
```

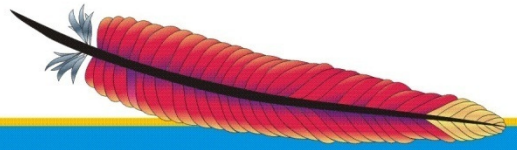
OSGi

- OSGi solves many common problems
  - Classloader hell
  - Dynamic systems
  - Updates and management of an installation
- Use frameworks/tools on top of OSGi
  - SCR, Spring-OSGi, Apache Sling

ApacheCon

The Future

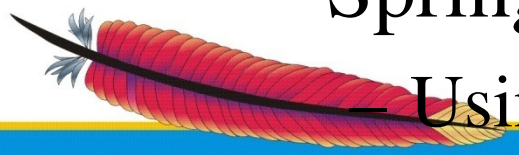
SPRING AND APACHE SLING



Leading the Wave
of Open Source

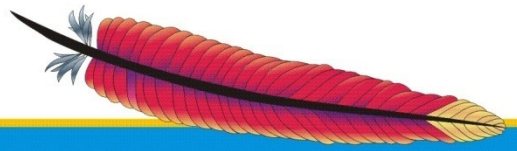
The Spring Framework

- Set of frameworks, libraries and tools
- The traditional Spring Container
  - API for registering services
  - XML configuration layer
  - Simplified Java layer
  - (Avalon to Spring Bridge)
- Spring-OSGi subproject
  - Using Spring inside an OSGi bundle



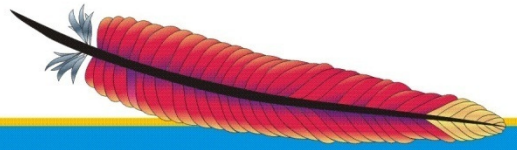
Apache Sling (Incubator)

- Web application framework
  - based on REST principles
  - content-oriented applications (through JCR)
  - runs in an OSGi environment
- Layered in bundles
  - separation of concerns



ApacheCon

SUMMARY



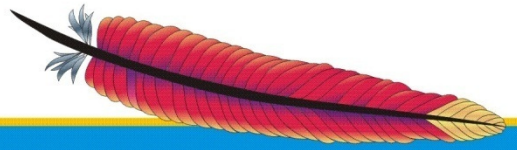
Leading the Wave
of Open Source

Summary

- Component oriented programming
  - Managing complex systems
  - Allows loose coupling
- OSGi
  - Dynamic systems
  - Updates and management of an installation
- Use frameworks/tools on top of OSGi
  - SCR, Spring-OSGi, Apache Sling

ApacheCon

QUESTIONS?



Leading the Wave
of Open Source