

Apache Container

Henning P. Schmiedehausen

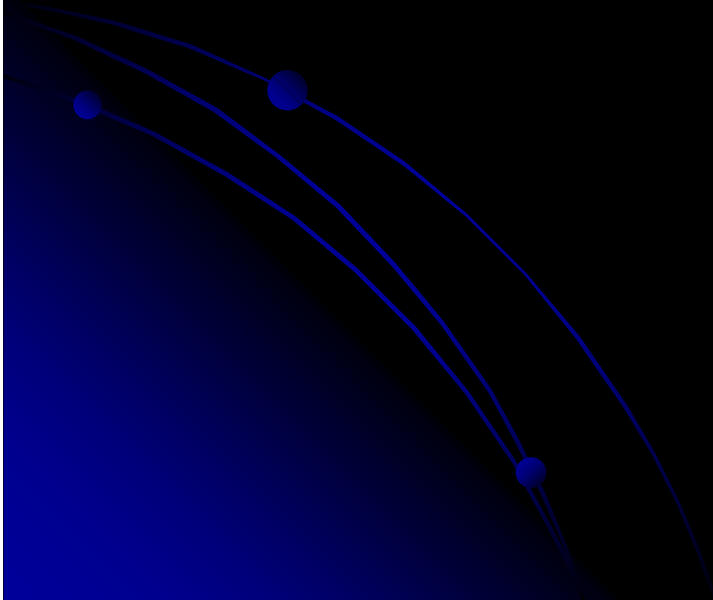
INTERMETA – Gesellschaft für Mehrwertdienste mbH

ApacheCon US 2005, December 12th 2005
San Diego, California, U.S.A.



Agenda

- Overview
- Container Concepts
- ASF Containers by example



EJB containers

- EJB containers are well known
 - heavy-weight
 - part of J2EE application servers
 - manages Session, Entity and Message driven beans
- OpenEJB used e.g. by Apache Geronimo
- This talk is about light-weight containers

Container definition

A Container...

- [...] contains a framework to help you create your own avalon containers. It boasts asynchronous management of your component instances, high scalability, easier main-tenance of your code, and easy embedding into various environments like servlet engines.
– **Apache Fortress**
- [...] is a lightweight and highly embeddable container for components that honour Dependency Injection.
– **Picocontainer**
- [...] is a light-weight implementation of a service framework using the Avalon service lifecycle interfaces aka Avalon container. – **Apache Turbine / YAAFI**
- [...] is a layered J2EE microkernel framework that enables the developers to use some ready-made functionality in their application code in a pluggable manner.
– **Spring Framework**
- [...] is a framework for creating applications, not an application, or even an application server, itself.
– **Apache HiveMind**
- [...] is a container that provides comprehensive support for the management of complex component based systems based on an underlying meta-model that facilitates automated assembly and deployment of components. – **DPML Metro**
- [...] delivers an open, common architecture for service providers, developers, software vendors, gateway operators and equipment vendors to develop, deploy and manage services in a coordinated fashion.
– **Apache Felix / OSGi**

→ Many definitions for one type of software!

Container definition

“A runtime entity that provides services to specialized Java components. Services provided include life cycle management, security, deployment and component-specific services for EJB, Web, JSP, servlet, applet and application clients.”

Kevin Taylor, Focus on Java

<http://java.about.com/cs/javadocumentation/q/container.htm>

Container definition

“A thing that does something for someone”

Henning Schmiedehausen

ApacheCon US 2005



Motivation

- Growing frustration with heavy-weight EJB containers
- Non-EJB frameworks became popular
- 2003: “Year of the container”
 - Spring - (www.springframework.org)
 - Pico – (www.picocontainer.org)
 - Avalon wars – led to Excalibur/Metro split
 - Eclipse moved to OSGi architecture

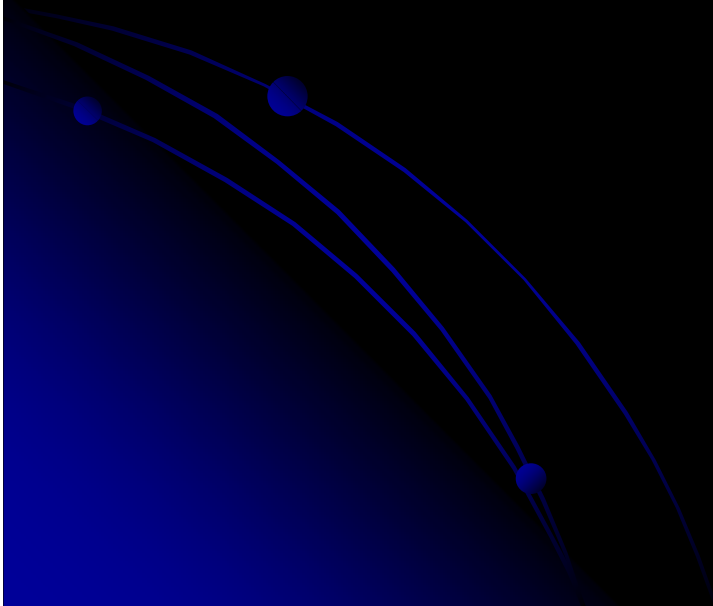
Containers and the ASF

- ASF has a long container history
 - Turbine offers a Service Layer since 2000
 - Avalon goes back to 1997/1998 (as “java-framework”)
- The “container” term was not yet popular
 - Avalon: Cocoon and James
 - Turbine: Jetspeed 1

ASF Containers today

- Apache Excalibur (TLP since 2004)
- Jakarta Hivemind (2004)
- Jakarta YAAFI (2004)
- Apache Felix (Oscar), in incubation (2005)

Container Concepts



Light-weight containers

- Popular term for “component manager”
- Related to component based development
- Often related to...
 - IoC (“Inversion of Control”)
 - Dependency Injection
 - Lifecycle Management
- Often responsible for...
 - Component instantiation
 - Component configuration
 - “Wiring up”

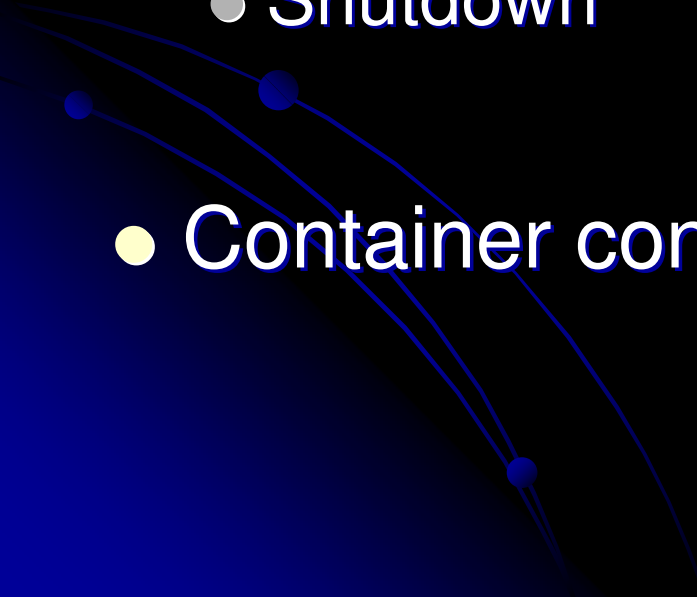
Inversion of Control (IoC)

- “term beaten to death”
- breaking dependencies between classes
- introducing interfaces for components
- components request dependencies from the container
- (for fun, look up the definition in Wikipedia)
(http://en.wikipedia.org/wiki/Inversion_of_control)

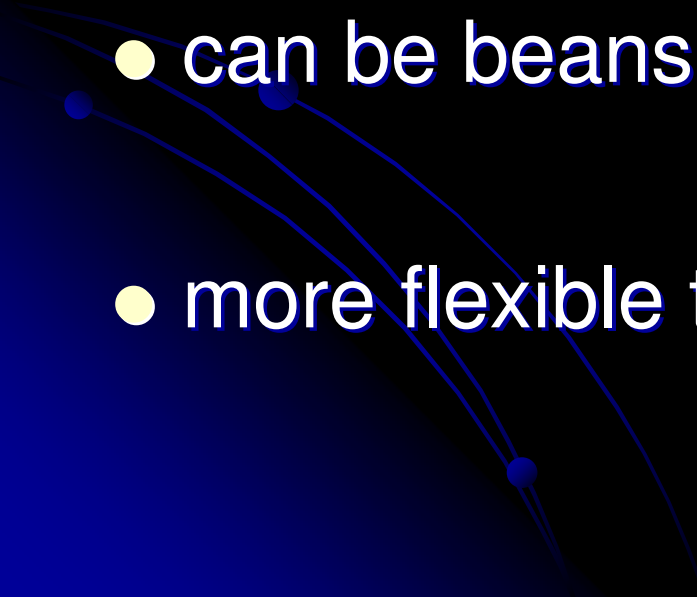
Dependency injection

- Components don't look for dependencies
 - Setter injection
 - Constructor injection
- Dependencies are configured, not hard-coded
- Container manages dependencies

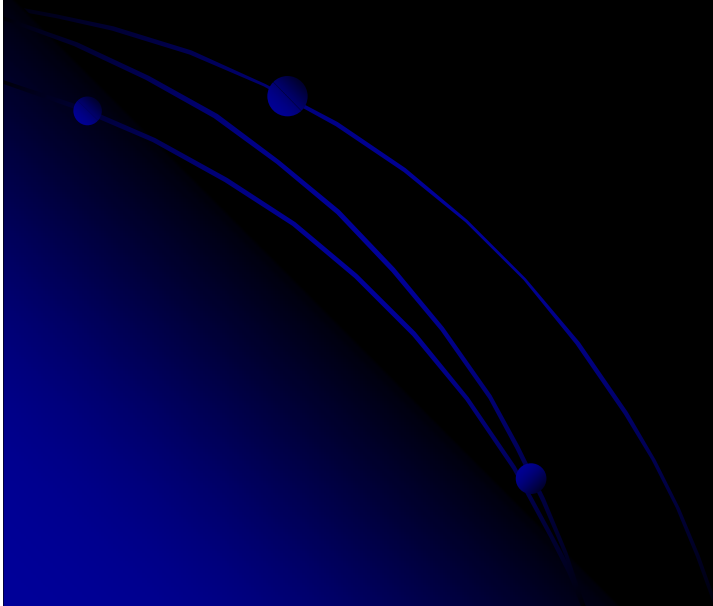
Lifecycle management

- Component requirements
 - Initialization
 - Configuration
 - Startup
 - Shutdown
 - Container controls lifecycle
- 

POJO

- „plain old java object“
 - objects don't need to implement interfaces
 - can be beans or arbitrary objects
 - more flexible than specific components
- 

Apache Containers by example



Factorial calculation

```
public class Demo {
    public void execute() {
        for (int i = 1; i <= 10; i++) {
            System.out.println("fac(" + getText(i)
                + ") = " + getText(calculate(i)));
            Thread.sleep(1 * 1000);
        }
    }

    private int calculate(final int n) {
        return (n == 0) ? 1 : n * calculate(n - 1);
    }

    private String getText(final int value) {
        [...convert number to text ...]
    }

    public static void main(String[] args) {
        new Demo().execute();
    }
}
```

Factorial calculation

fac(one) = one

fac(two) = two

fac(three) = six

fac(four) = two-four

fac(five) = one-two-zero

fac(six) = seven-two-zero

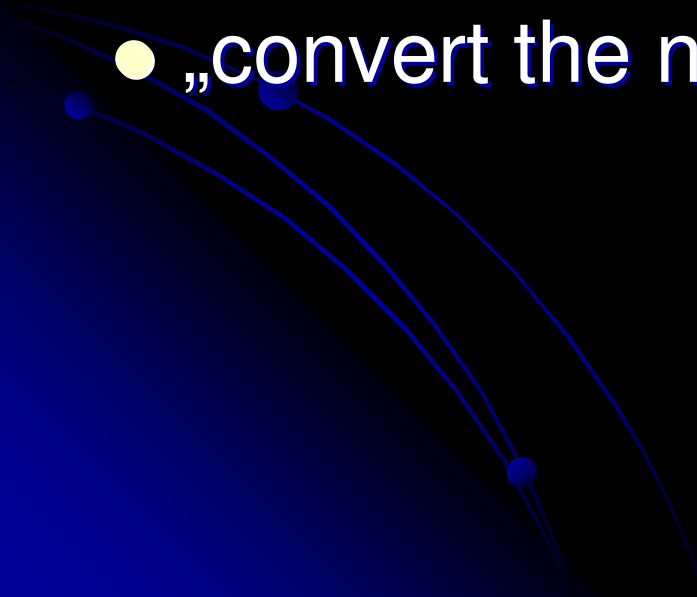
fac(seven) = five-zero-four-zero

fac(eight) = four-zero-three-two-zero

fac(nine) = three-six-two-eight-eight-zero

fac(one-zero) = three-six-two-eight-eight-zero-zero

Splitting into components

- „generating the number“
 - „calculating the factorial“
 - „convert the number into String“
- 

Components without Container

Initialize Components by hand

```
public class Demo
{
    private final NumberGenerator ng;
    private final Factorizer fc;
    private final NumberConverter nc;

    private int endValue = 0;

    public Demo() {
        ng = new NumberGenerator();
        fc = new Factorizer();
        nc = new NumberConverter();
    }

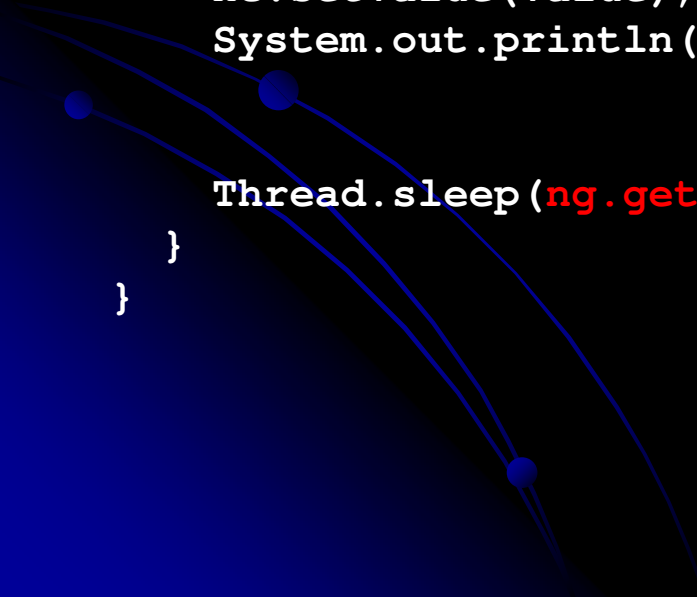
    public void init(final int startValue, final int endValue, final int delay) {
        ng.setStartValue(startValue);
        ng.setDelay(delay);
        this.endValue = endValue;
    }
}
```

„business logic“

```
public void execute()
{
    while (ng.getCurrentValue() <= endValue)
    {
        int value = ng.getNextValue();
        fc.setValue(value);
        nc.setValue(fc.getFactorial());
        String res = nc.getText();

        nc.setValue(value);
        System.out.println("fac(" + nc.getText() + ") = "
            + nc.getText());

        Thread.sleep(ng.getDelay() * 1000);
    }
}
```



Apache Excalibur

- Avalon was the project “that started it all”
- Excalibur inherited most of the now closed Avalon project
- Avalon compatible containers
 - ECM (Excalibur Component Manager)
 - Fortress
 - 3rd party: Loom, Keel, Merlin, YAAFI

Avalon components


- Components must implement Avalon interfaces
 - e.g. Initializable, Startable, Configurable
- Lifecycle defined in a rigid sequence
- Enforces IoC, Separation of concerns
- Components are sometimes container specific

Excalibur Fortress

- Default container of Excalibur
- used e.g. in Apache Cocoon
- Uses Meta Tags in source:

```
/**
 * A number generator.
 *
 * @avalon.component
 * @avalon.service type=„NumberGeneratorAPI“
 * @x-avalon.info name=„NumberGenerator“
 * @x-avalon.lifestyle type=„singleton“
 */
public class NumberGenerator
    implements NumberGeneratorAPI {
    // ....
}
```

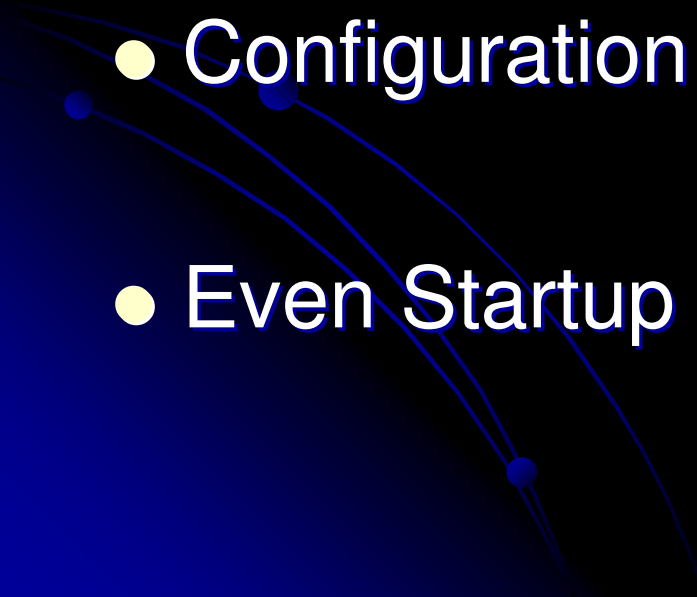
Excalibur Fortress

- Fortress (and Avalon!) consists of a number of API and Implementation jars
 - Documentation is a bit wanting
 - So getting „over the hump“ is not simple
- 

Hivemind

- „Service Microkernel“
- „Module deployment descriptor“ in XML
- Allows custom configuration schema
- Dynamic interceptors
- Dependency injection
(Constructor and Setter)
- Services can be POJOs

Using Hivemind

- Wiring is done by Hivemind
 - Components are injected through setters
 - Configuration also injected
 - Even Startup controlled by Hivemind
- 

Hivemind Configuration

```
<module id="container.talk" version="1.0.0"  
  package="de.intermeta.container.hivemind">
```

```
  <service-point id="Demo" interface="impl.DemoImpl">
```

```
    <invoke-factory>
```

```
      <construct class="impl.DemoImpl">
```

```
        <set property="endValue" value="10" />
```

```
      </construct>
```

```
    </invoke-factory>
```

```
  </service-point>
```

```
[...]
```

```
<service-point id="NumberConverter"
```

```
  interface="de.intermeta.components.NumberConverter">
```

```
    <create-instance class="de.intermeta.components.NumberConverter"/>
```

```
  </service-point>
```

```
[...]
```

```
</module>
```

YAAFI

- „Yet Another Avalon Framework Implementation“
- Lightweight, depending only on Avalon Framework
- can emulate Fortress, ECM, Phoenix, Merlin

- Dynamic Proxies and Interceptors
- Reconfiguration „on the fly“ possible

- Developed as Part of Turbine 3
<http://jakarta.apache.org/turbine/fulcrum/fulcrum-yaafi/>

Using YAAFI

```
public class DemoImpl extends AbstractLogEnabled
    implements Executable, Configurable, Serviceable
{
    public void configure(Configuration cfg)
        throws ConfigurationException
    {
        setEndValue(cfg.getChild("end").getValueAsInteger(10));
    }

    public void service(ServiceManager serviceManager)
        throws ServiceException
    {
        nc = (NumberConverterAPI) serviceManager.lookup(
            "de.intermeta.container.yaafi.api.NumberConverterAPI");
        ng = (NumberGeneratorAPI) serviceManager.lookup(
            "de.intermeta.container.yaafi.api.NumberGeneratorAPI");
        fc = (FactorizerAPI) serviceManager.lookup(
            "de.intermeta.container.yaafi.api.FactorizerAPI");
    }
}
```

YAAFI Config

- Very similar to the „old ECM configuration“
- Role and component configuration file loaded by the container
- Role file matches API and implementation class
- Component file for component configuration

```
<role-list>  
  <role name="de.intermeta.container.yaafi.api.NumberGeneratorAPI"  
    shorthand="NumberGenerator"  
    default-  
    class="de.intermeta.container.yaafi.impl.NumberGeneratorImpl"/>  
</role-list>
```

YAAFI Config

```
<componentConfig>  
  <NumberGenerator>  
    <start>1</start>  
    <delay>1</delay>  
  </NumberGenerator>  
</componentConfig>
```

```
[...]  
public void configure(Configuration c)  
    throws ConfigurationException {  
    setStartValue(c.getChild("start").getValueAsInteger(1));  
    setDelay(c.getChild("delay").getValueAsInteger(1));  
}  
[...]
```

Apache Felix

- „OSGi R4 compliant container“
- Still in incubation
- ASLv2 licensed implementation of the OSGi spec
- Other OSGi implementations:
 - Knopflerfish, Oscar, Eclipse

Apache Felix

- „The OSGi Alliance and its members specify, create, advance, and promote wide industry adoption of an open delivery and management platform for application services in home, commercial buildings, automotive and industrial environments.“
- Industry standard with compliance tests

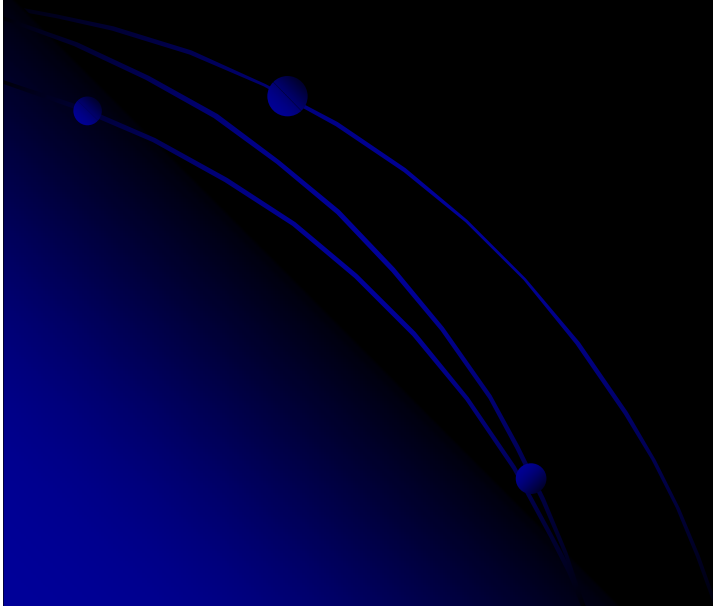
Apache Felix



Summary

- The ASF offers a number of Containers
 - Mature: Excalibur (Fortress), Hivemind
 - Usable: YAAFI, (ECS)
 - Under development: Felix
- You should evaluate before deciding!

Any questions?



Where to go from here

- <http://henning.schmiedehausen.org/container/>
 - This talk and the example code to test
- Container Homepages
 - <http://jakarta.apache.org/hivemind/>
 - <http://excalibur.apache.org/>
 - <http://jakarta.apache.org/turbine/fulcrum/fulcrum-yaafi/index.html>
 - <http://incubator.apache.org/felix/>

Where to go @ ApacheCon

- WE12 – Wednesday, 14:00
 - „The future of Open-Source OSGi“
- WE16 – Wednesday, 15:00
 - „Java Modularity Support in OSGi R4“
- WE20 – Wednesday, 16:30
 - „Declarative Services in OSGi R4“

Thanks for your attention!

