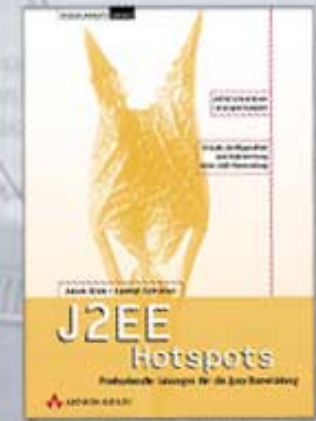


SOA with Beehive

Enterprise Architect

“THE ARCHITECTURE SHOULD BE MADE AS SIMPLE AS POSSIBLE, BUT NOT SIMPLER”

**ALBERT EINSTEIN,
DECORATED BY ADAM BIEN**



www.adam-bien.com

Who am I?

Adam Bien (www.adam-bien.com)

- **BEA Technical Director, freelancer and book author:**
 - Enterprise Java Frameworks, Addison Wesley
 - J2EE Patterns, Addison Wesley
 - J2EE HotSpots, Addison Wesley
 - Struts, Java Framework für Webanwendungen, Java Magazin
- **Java Trainer (since Java 1.0).**
 - Sun Microsystems
 - oose.de
- **Published articles @ „Java Magazin“**
 - JavaMagazin April/2004, „**Component Based Development and JSR 175**“
 - JavaMagazin April/2004-November, „**Principles of Architecture**“
 - JavaMagazin „**J2EE Patterns**“



What is Beehive?

What is Beehive?

- Beehive is a project that BEA proposed to the Apache Software foundation (incubator)
- Beehive consists of Java Web Services, NetUI PageFlows and Controls
- Beehive uses JSR-175 and JSR-181 to define Meta Data annotations

What is Beehive? (developer view 😊)

- Beehive is a revival of JavaBeans
- Using defined metadata higher degree of automatation can be established
- Beehive controls (JavaBeans) are executed in Java Bean Runtime (based on JavaBeans Runtime Containment and Services Protocol.)

The Beehive Components

- XMLBeans –Java-XML binding tool through which you can access the full power of XML in a Java-friendly way.
- Java Controls - JavaBean-based component model for resource access
- Java Web Services - Annotations for rapid Web Service development
- Java Page Flows - Struts-based MVC framework for Web application design

XMLBeans: a Beehive Building Block

XMLBeans provide a familiar, JavaBean-based view of XML data

- Without losing access to the original, native XML structure
- Beans are bound to the XML document...not imported/exported

XML Schema used to compile strongly-typed Java interfaces and classes

- XMLBeans supports all XML schema definitions
- Full XML Infoset fidelity
- Reflect into the XML schema itself through an XML Schema Object model

XMLBeans access to XML is fast and rich:

- Strong-typed getters and setters
- XQuery transformations
- Loosely-typed via cursors

Separate Apache Project will be bundled in Beehive distribution

XML Beans



XMLBeans: a Beehive Building Block

```
CustomerDoc doc = customerDocument.Factory.newInstance();  
Customer c = doc.addNewCustomer();  
ad.setName("duke");  
ad.setAddress1("java road");  
ad.setCity("j2ee city");
```



Java Controls

Java Controls Features

- Access business logic or a resource in a consistent, straightforward manner as a POJO
- Can be extensible, meaning that the control can be customized via annotations (JSR-175)
- May be used from a variety of clients
- Enable more people to be successful at developing Java
- Makes J2EE experts significantly more productive
- Helps experts by encapsulating their expertise into controls that can be used simply by others

Java Controls

Provide a consistent model for discovery of:

- Resource operations (actors)
- Resource events (notifications)
- Configuration options (properties)

Client programming model based upon a JavaBean type.

- JavaBean wrapper around Control Implementation
 - Per-instance storage of dynamic properties
 - Property resolution services
 - Event listener routing
 - Mgmt of contextual services and nested Controls

Java Controls

Transparent resource management

- Acquisition mechanisms and guarantee of resource release
- Cooperates with outer container to manage resources

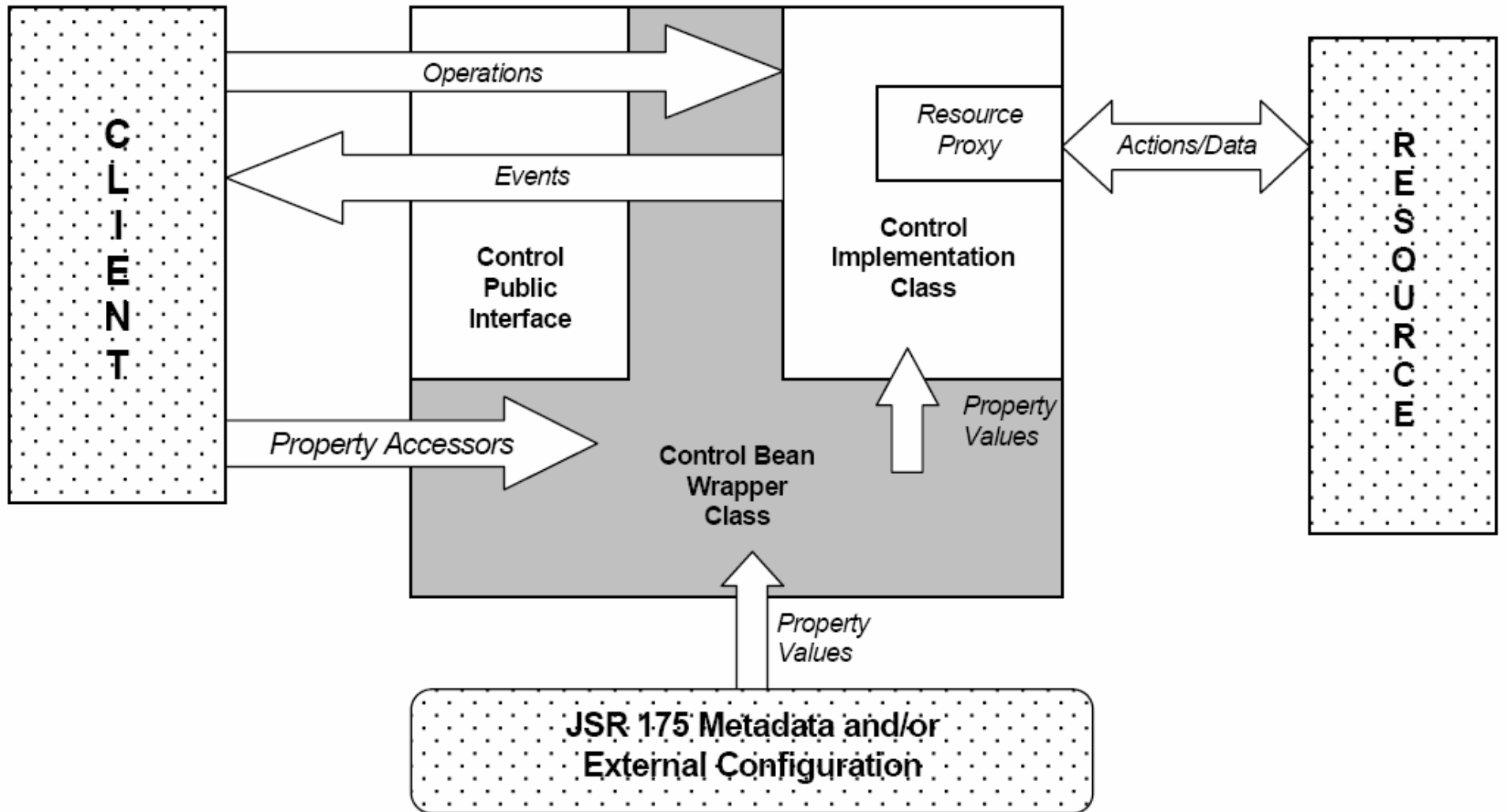
Extensibility Model

- Define operations for specific resource use cases
- Customized resource facades using metadata

Focus developer on “what”, not “how”

- Make mechanisms less visible!
- Simple things should be simple!

Java Control Architecture



Java Control Example (JMS)

```
@ControlInterface public interface JmsMessageControl {
    // OPERATIONS
    public void sendTextMessage(String text);
    public void sendObjectMessage(java.io.Serializable object);

    // EVENTS
    @EventSet public interface Callback {
        public void onMessage(javax.jms.Message msg); }

    //PROPERTIES
    @PropertySet @Target({ElementType.FIELD, ElementType.TYPE})
    public @interface Connection {
        public String factoryName();}
}
```

Java Controls

Control Public Interface:

- Source file that defines the set of operations, events, extensibility model, and properties associated with the Control type.

Control Implementation Class:

- Source file that provides the implementation of the operations and extensibility model described by the *Control Public Interface*.

ControlBean Generated Class:

- Code-generated JavaBean class that is derived from the Control Public Interface and the Control Implementation Class by a Control compiler.



Java Web Services

JSR-181

- JSR-181 defines annotations to enable easier Web service programming. Example:

```
@WebService public class TaxCalculator {  
    @WebMethod public float computeTax(String productId) { }  
}
```

- Abstracts SOAP marshalling, Java-to-XML binding, WSDL file creation, underlying bean deployment, and much more
- Uses the J2EE 1.4 WebService infrastructure (JAX-RPC 1.1) and JSR-175

JSR-181

- Provides a defined set of JSR-175 Tags
- Generates WebService infrastructure for annotated code
- Enables “linking” of WSDL 1.1 definitions with existing code
- Service implementations should adhere to the JavaBean standard



Java Page Flows

Java Page Flows

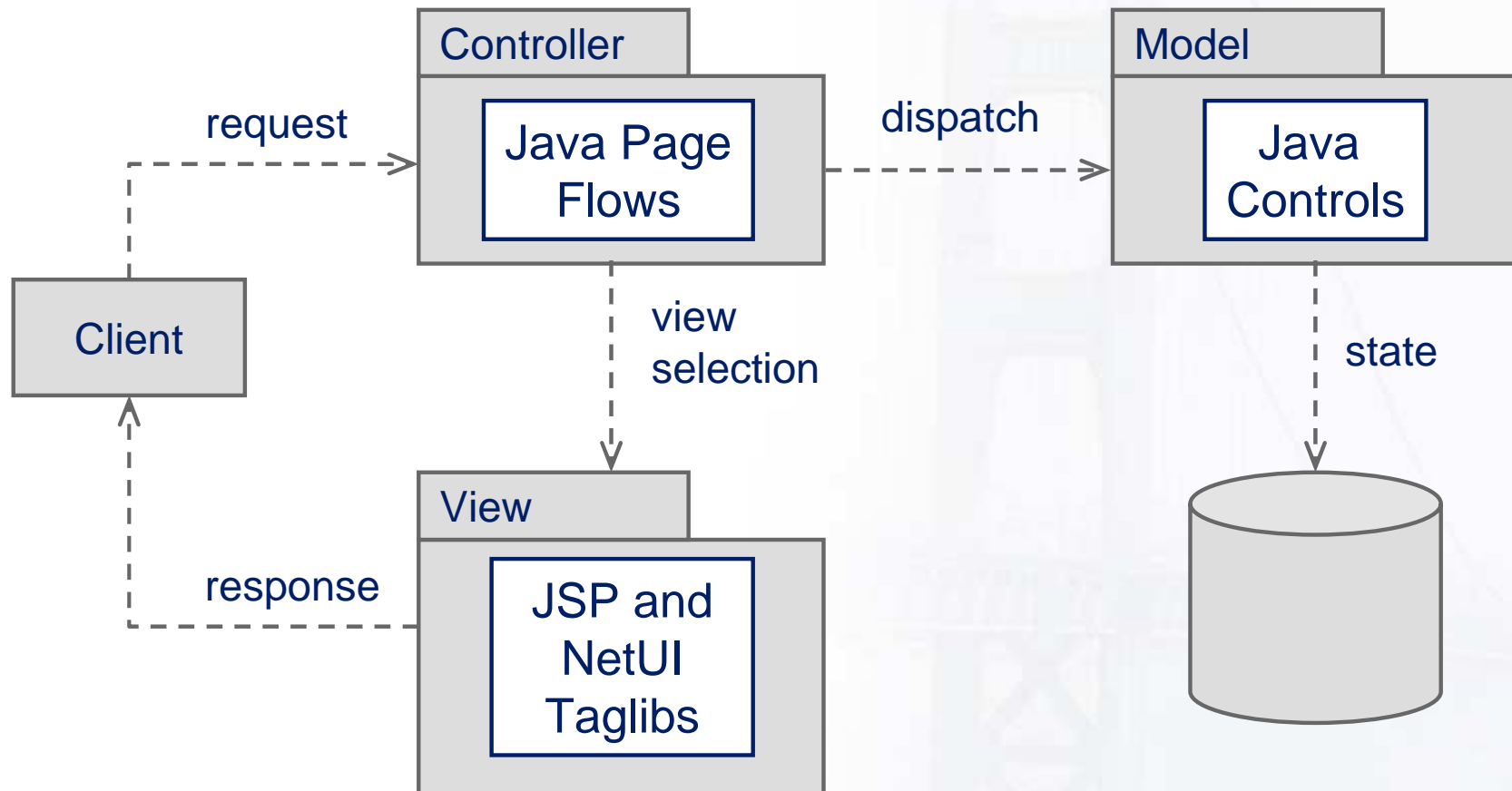
What are Java Page Flows?

- MVC web application framework built on Apache Struts
- Manages the presentation and interaction flow for your web application
- Centralizes everything related to the Controller.
- Union of code and metadata (JSR 175)
- Supports Java Server Faces for a view

Basic elements

- Controller: page flow class
- Actions: annotated methods
- Exception Handlers: annotated methods
- Page Flow State: member fields

Java Page Flows Architecture



What is a component?

What is a component?

„A software component is a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third parties.“

[C. Szyperski Component Software – Beyond Object-Oriented Programming]

CBD Concepts

- Separation Of Concerns
- Layering (strict oder loose)
- Private vs. public views
- Specification vs. Realization
- Viewpoints

Business components

- Are divided into a business interface and the realization.
- The realization could be implemented using well defined patterns or approaches.
- The business interface should be coarse grained, to minimize the chattiness.

Business components

- A components needs a defined runtime environment.
- A component is remote accessible.
- The runtime environment provides additional infrastructure or pervasive services.

Business components – the pragmatic viewpoint

- A component is a package, subsystem or UML-component.
- Cohesive classes are grouped in a container (package, subsystem or component) which should be independent to its neighbors.
- Fine grained and therefore reusable classes will be coordinated by an Facade.

Business components – the pragmatic viewpoint

- The Facade is often decorated by a Decorator (GoF).
- The Facade implements the component's business interface.
- A component is able to implement several interfaces.

Business components – the pragmatic viewpoint

- Are only accessible through the public interface.
- Are made of fine grained objects which realize the business interface.
- Need a predefined infrastructure like a EJB- or .NET container.

Business components – a procedural approach

- Because of Separation Of Concerns, the business logic is separated from the state.
- The realization is no more object oriented, but **procedural**.
- Inheritance is hard to implement in a procedural world.

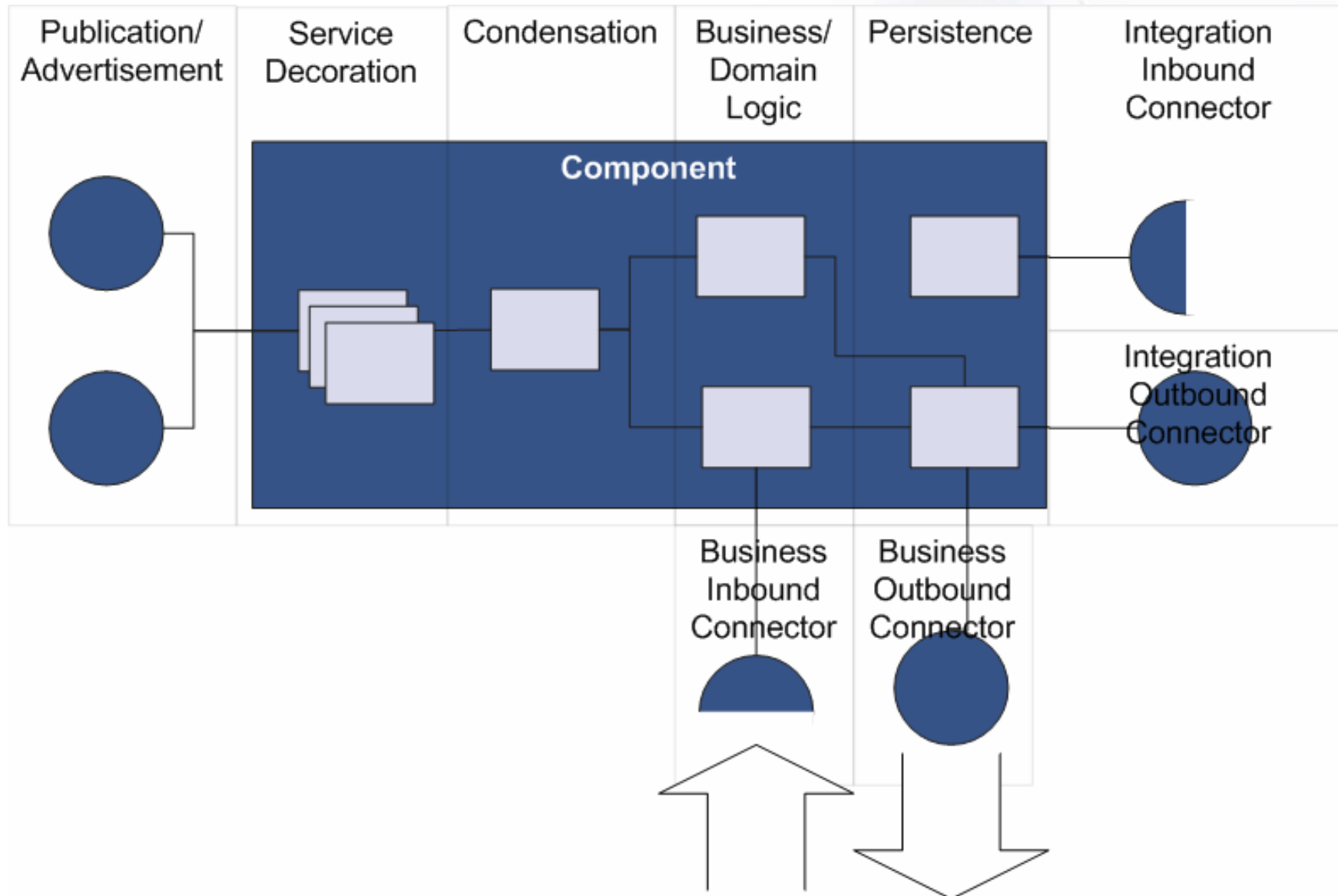
CBD – die Erklärung

- A business component could be compared with a macro objects.
- Usual POJOs are too fine grained to be accessed remotely.
- The business interface is comparable with the public methods of an object, and the component realization is similar to the object's private methods.

Business Components

- It is not suitable in the first development iterations to deal with “low level” implementation or J2EE deployment-settings.
- Abstraction of the concrete implementation increases portability, reduces migration costs and simplifies the design process of building J2EE-applications
- Technology independent models can be reused in case the technology platform changes.

Business Components



Business components

Layer responsibilities:

- **Integration:** encapsulates incompatible resources
- **Persistence:** organizes the component's entities
- **Business/Domain Logic:** packages the component's business logic and algorithm implementation
- **Condensation:** aggregates fine grained elements into coarse grained service interfaces
- **Service Decoration:** provides additional decoration layers like: transaction, security or conversational state.
- **Publication/Advertisement:** provides a clean viewpoint to the component's services. Describes the functionality of the components with preconditions, postconditions and invariants.

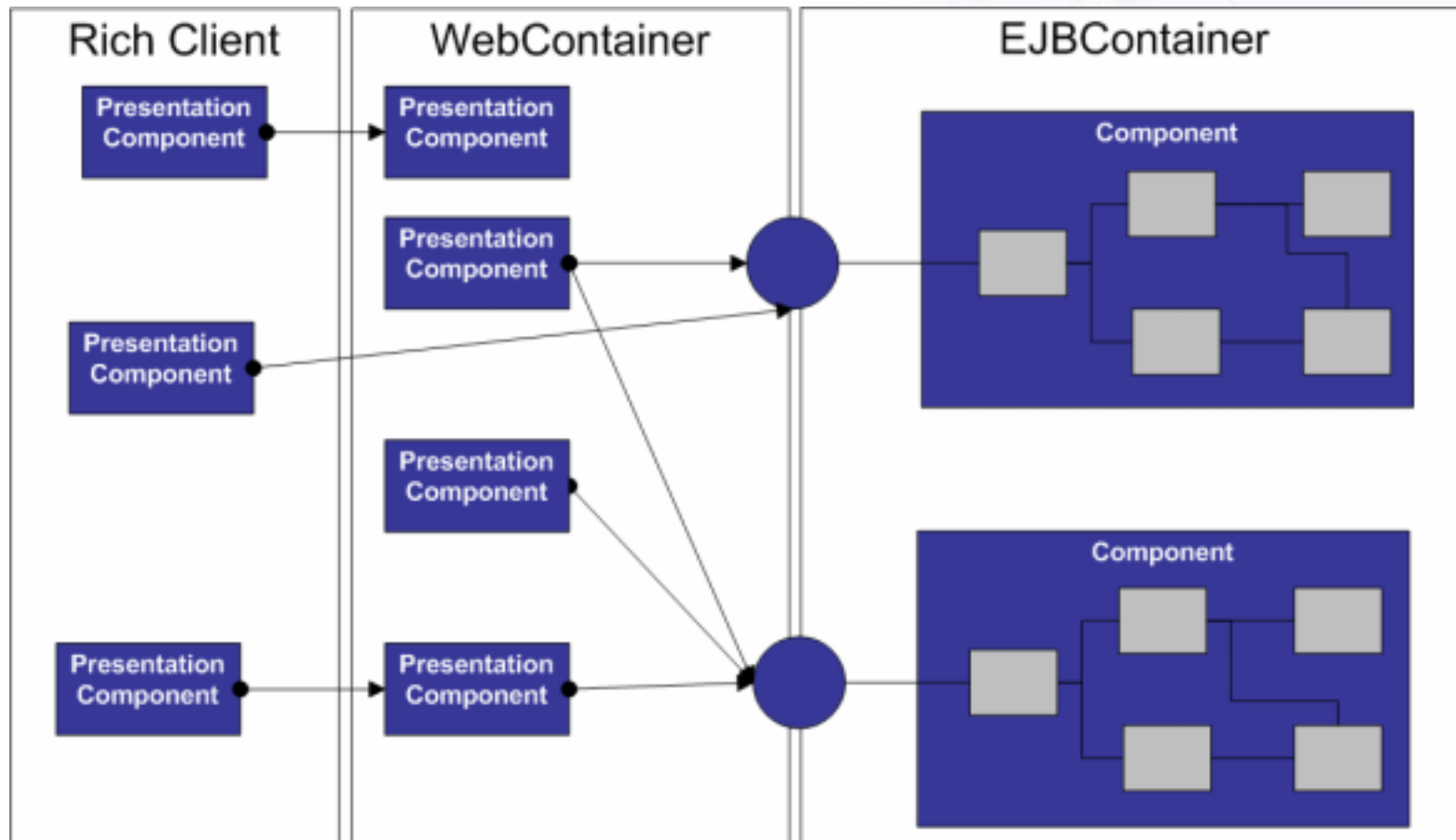


CBD/SOA with Beehive

Beehive from the architecture perspective

- A meta architecture (e.g. layers, patterns etc.) or an architecture template defines the outline for J2EE projects without coupling to given technology or even Beehive itself.
- Beehive can be used for the instantiation of such an architecture. Especially the separation between the technical aspects and the realization of the business needs could be easier established.

Beehive from the architecture perspective



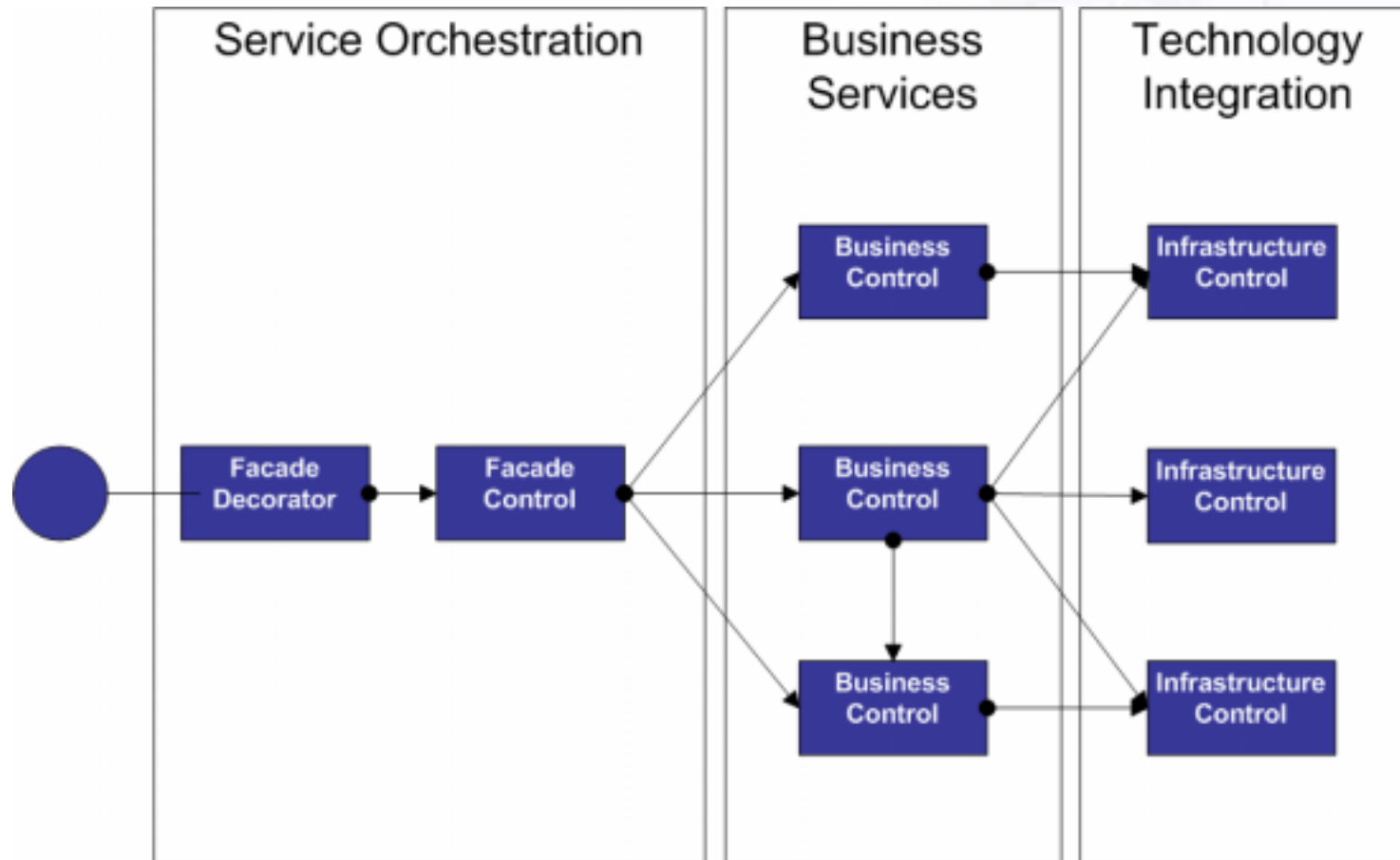
CBD/SOA with Beehive

- Beehive provides a metadata driven programming model for components at different layers of abstraction.
- Fine grained and therefore more reusable components can be composed to more valuable services.
- An already tested infrastructure could be used as a runtime for coarser components

Beehive in different iteration cycles

1. Fine grained infrastructure services represents the technical foundation for components (connection pools, connectors, event controllers etc.). Especially the persistence and integration layer could benefit from a higher degree of automation.
2. Already defined and tested infrastructure components could be used for orchestration purposes.

Beehive in different iteration cycles



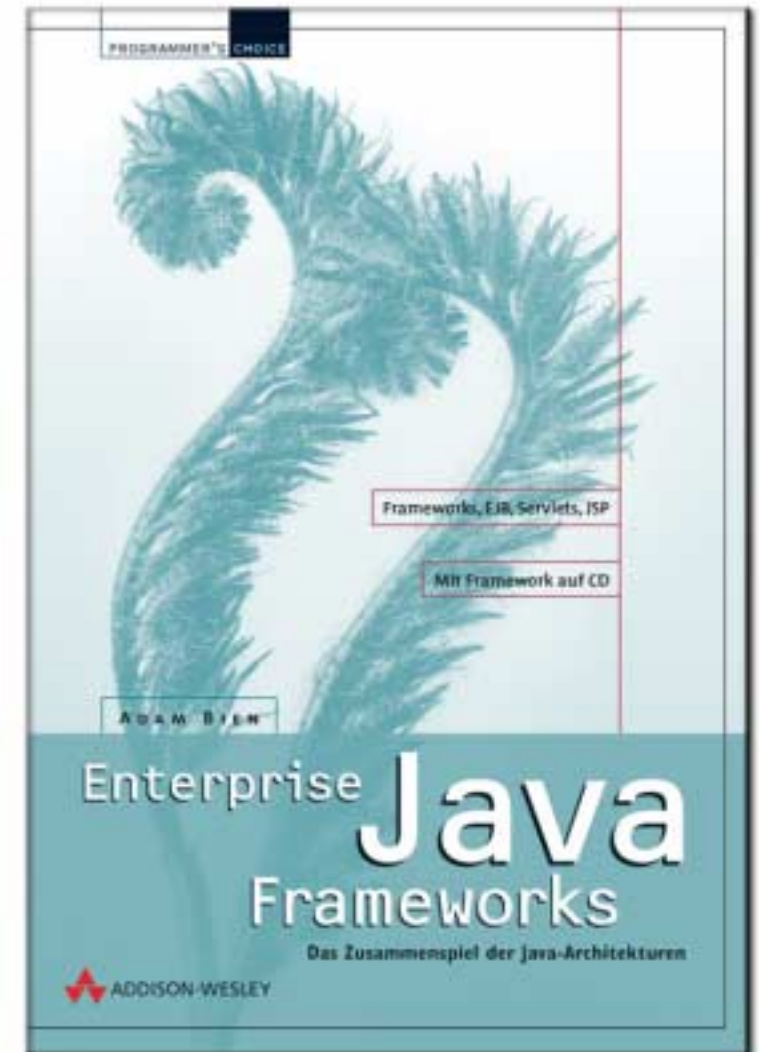
Beehive in different iteration cycles

2. Façade Controllers are responsible for coordination and implementation of additional functionality.
3. Façade Controllers are decorated with additional technical services e.g auditing, transactions, remoting (JSR-181)
4. All layers together represent a SOA/Beehive component which fulfills the functional requirements and therefore the Use Cases.

References

Referenzen

- <http://www.adam-bien.com>
- Enterprise Java Frameworks, Adam Bien, Addison Wesley, ISBN 3-8273-1777-0
- J2EE Patterns, Adam Bien, Addison Wesley, ISBN 3-8273-1903-X
- Core J2EE Patterns, John Crupi, Deepak Alur, Dan Malks, Prentice Hall PTR, ISBN 0-1306-4884-1
- Entwurfsmuster . Elemente wiederverwendbarer objektorientierter Software, Gamma, Helm, Johson, Vli, Marchand, Muzyka, Addison Wesley, ISBN 3-8273-1862-9



Referenzen

- <http://www.controlhaus.org>
- <http://incubator.apache.org/beehive/>
- <http://jcp.org/en/jsr/detail?id=181>

Referenzen

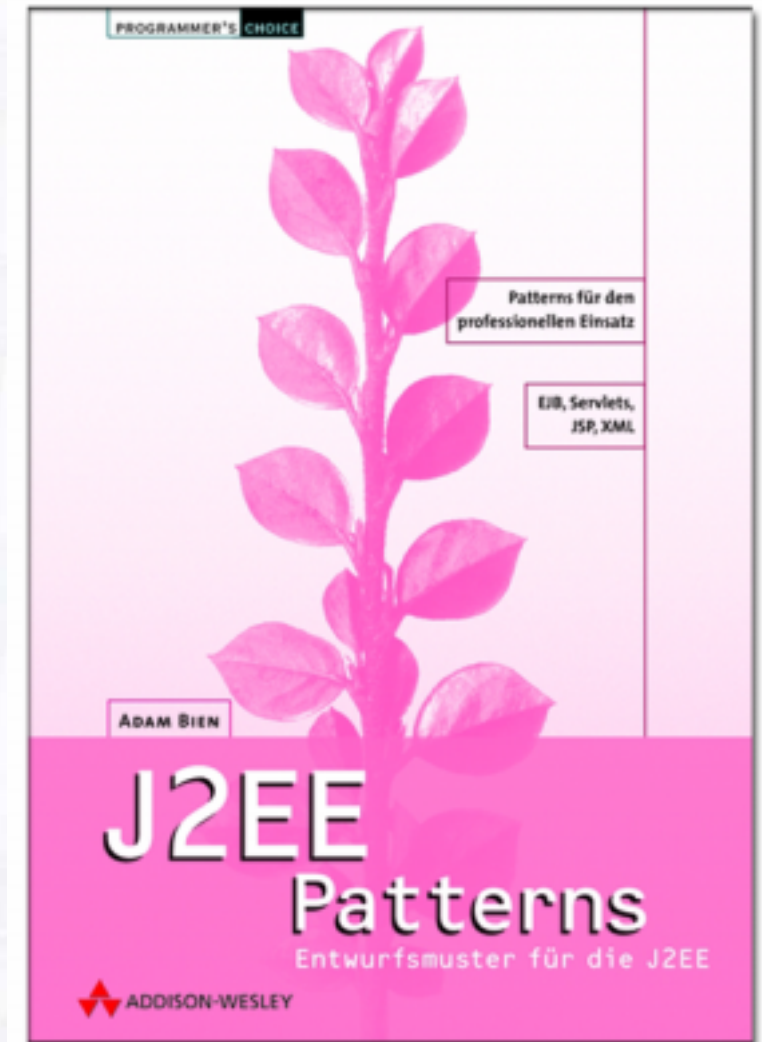
<http://www.borland.com>

<http://developer.java.sun.com/developer/restricted/patterns/J2EEMPatternsAtAGlance.html>

<http://www.sitraka.com>

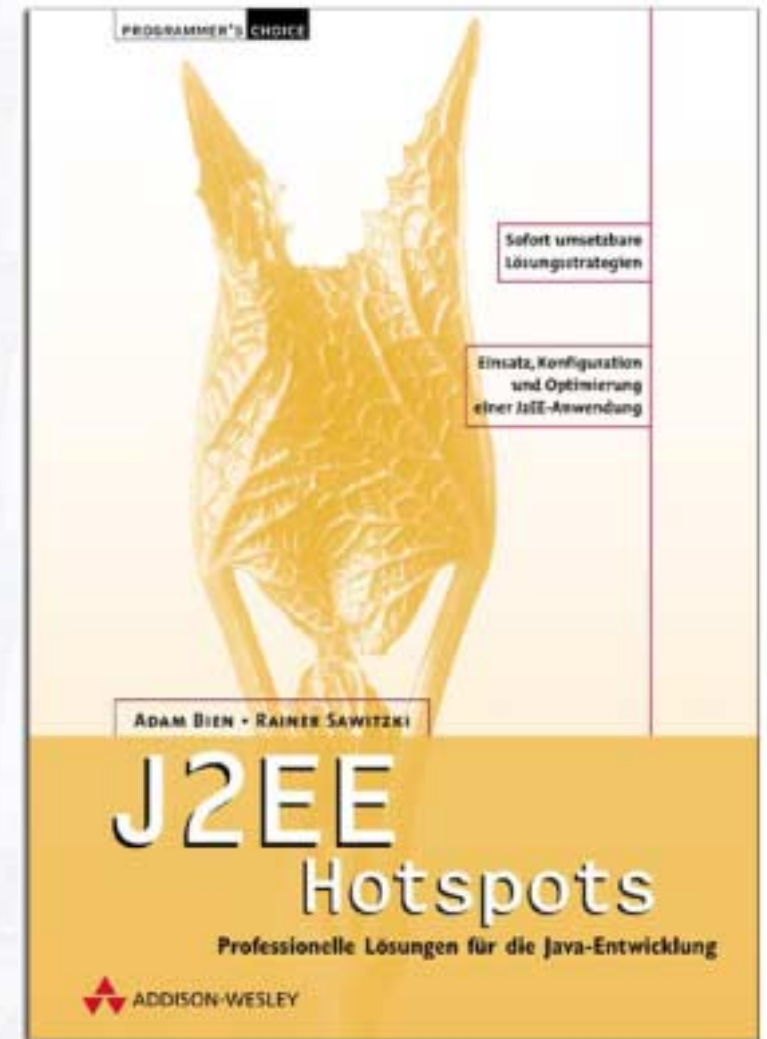
<http://www.bea.com>

<http://www.j2ee-patterns.com>



Referenzen

- JavaMagazin Dezember/2000, "Cache Me !" (Servlet programming)
- JavaMagazin Oktober/2001, "Dekorationsfabriken" (Patterns)
- JavaMagazin Juni/2002 „Die Performance des Activators“ (Service Activator mit WLS 7.0)
- JavaMagazin Oktober/2002 J2EE Patterns
- <http://www.jboss.org>
- <http://www.star-finder.com>



Interested in „highend“ trainings, coaching, project support?

...just drop me an email 😊

=> abien@adam-bien.com