Dr.-Ing. Michael Eichberg
eichberg@informatik.tu-darmstadt.de

TECHNISCHE UNIVERSITÄT DARMSTADT

The Interceptor Architectural Pattern

Pattern-oriented Software Architecture
Volume 2 Patterns for Concurrent and Networked Objects;
Douglas Schmidt, Michael Stal, Hans Rohnert and Frank Buschmann, Wiley 2000

Goal: Supporting a wide-range of Applications

(Naïve approaches)

Integration of all services

- Often infeasible, because not all (required) services can be anticipated
  (Integrating services later on typically complicates the design and maintenance.)
- Services that are not required still require resources (memory, processor cycles)

frameworks supporting a...

Do not provide any services

- Application developers that require services that are not available have to implement them on their own
  (They have to implement logic not related to the application domain.)
- A couple of services require a tight integration with the (component) framework

Related Design Principle:
Open-closed design principle
(open for extension, but closed for modifications)

Enabling Service Integration

Forces

- A framework should allow the **integration of additional services without requiring modifications** to its core architecture
- The integration of application-specific services into a framework should not ...
  - affect existing framework components
  - require changes to the design or implementation of existing applications
- Applications that use a framework may need to monitor and control its behavior

Enabling Service Integration

Solution

- Allow applications to **extend a framework transparently** by registering “out-of-band” services with the framework via predefined interfaces (*interceptor callback interfaces*)
- Trigger these services when “certain” events occur (... i.e., when application relevant events occur)

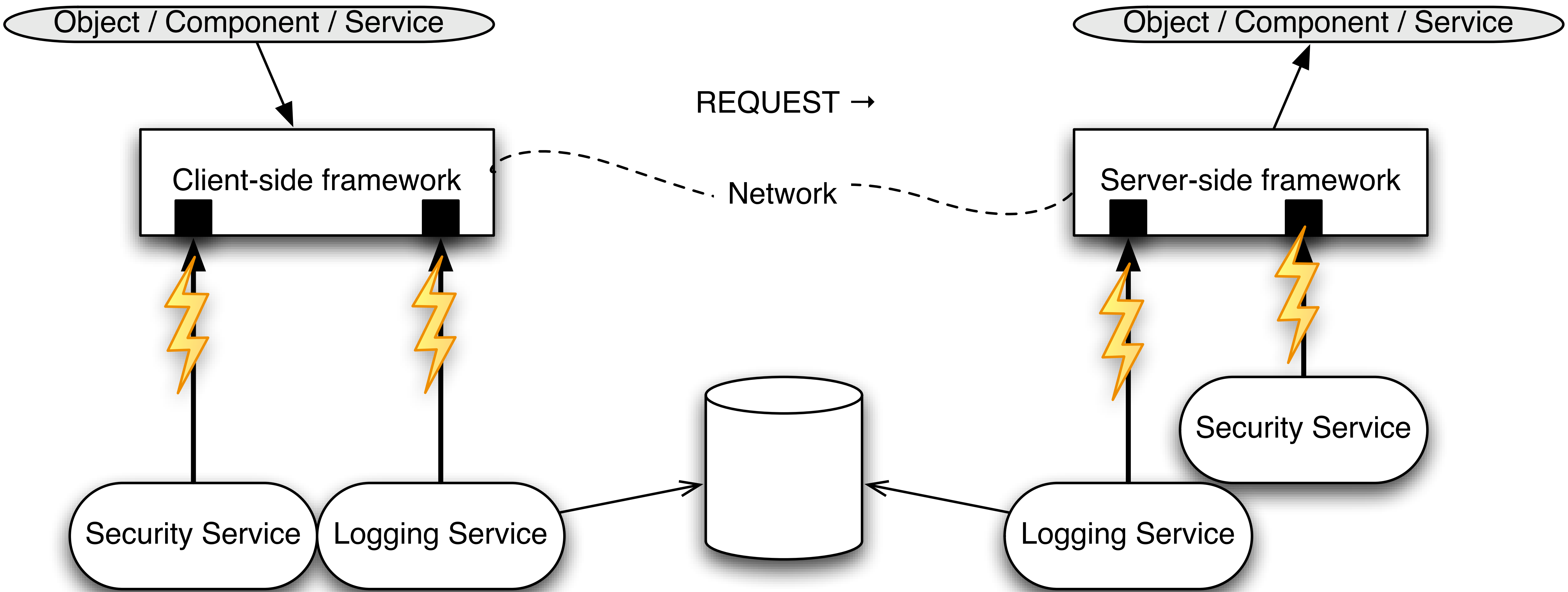
Intent

Purpose / Goal

examples are: logging, security, load balancing,...

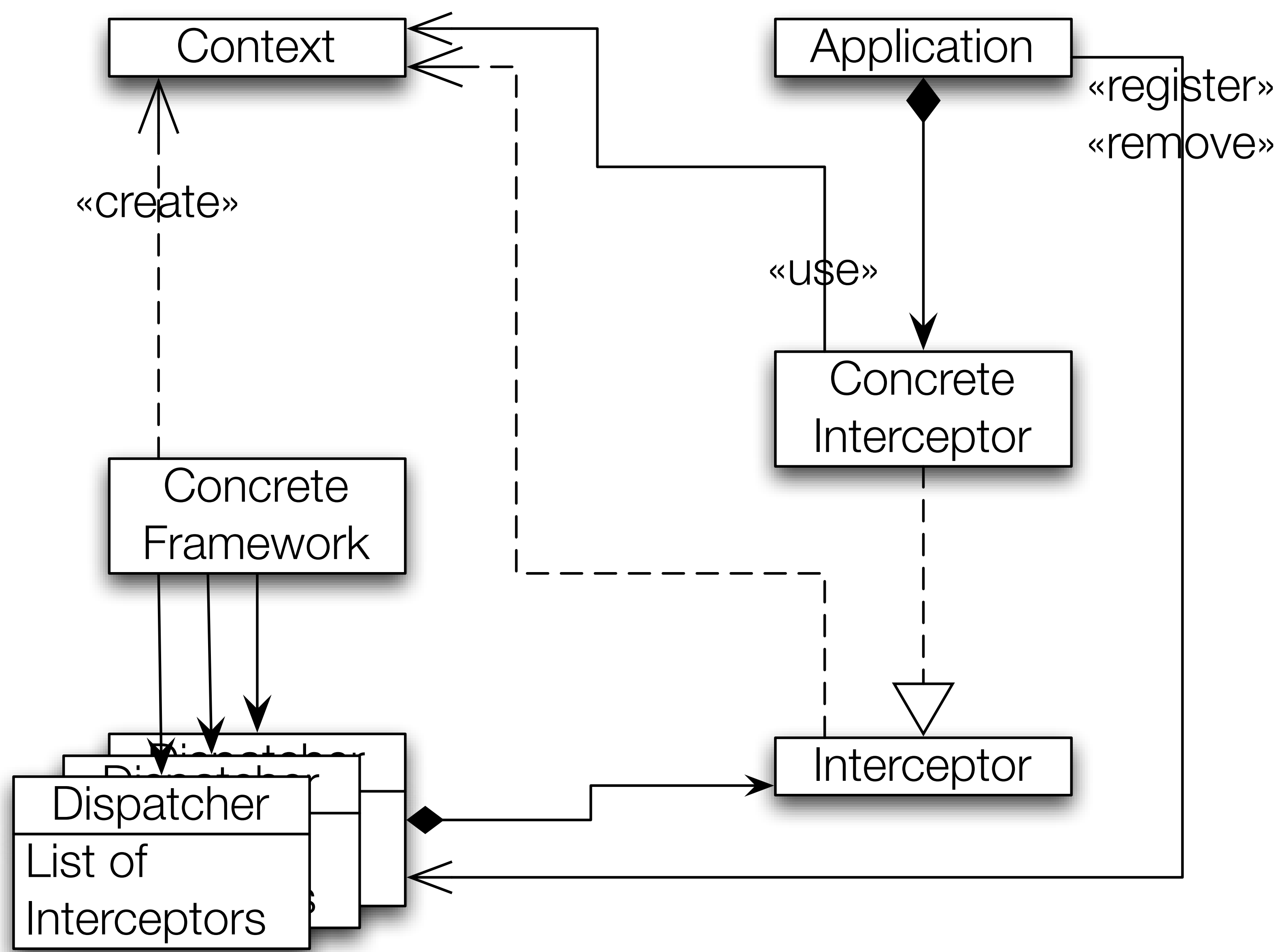
The interceptor architectural pattern allows **services to be added transparently** to a framework and **triggered automatically when certain events occur**.

Intercepting Events



[p. 109; Pattern-oriented Software Architecture Volume 2; D. Schmidt, M. Stal, H. Rohnert and F. Buschmann; Wiley 2000]

Collaborations



[p. 115; Pattern-oriented Software Architecture Volume 2; D. Schmidt, M. Stal, H. Rohnert and F. Buschmann; Wiley 2000]

Identifying and Assigning Responsibilities

Class	Collaborations
Concrete Framework	‣ Dispatcher
Responsibilities ‣ defines application services ‣ integrates dispatchers that allow applications to intercept events ‣ delegates events to associated dispatchers	

Identifying and Assigning Responsibilities

Class	Collaborations
Interceptor	- N/A
Responsibilities	
- defines an interface for integrating out-of-band services	

Identifying and Assigning Responsibilities

Class Concrete Interceptor	**Collaborations** - Context Object
Responsibilities - implements a specific out-of-band services - uses context-object to control the concrete framework	

Identifying and Assigning Responsibilities

Class	Collaborations
Dispatcher	‣ Interceptor ‣ Application
Responsibilities ‣ allows applications to register and remove concrete interceptors ‣ dispatches registered concrete interceptor callbacks when events occur	

Identifying and Assigning Responsibilities

Class	Collaborations
Context Object	‣ Concrete Framework

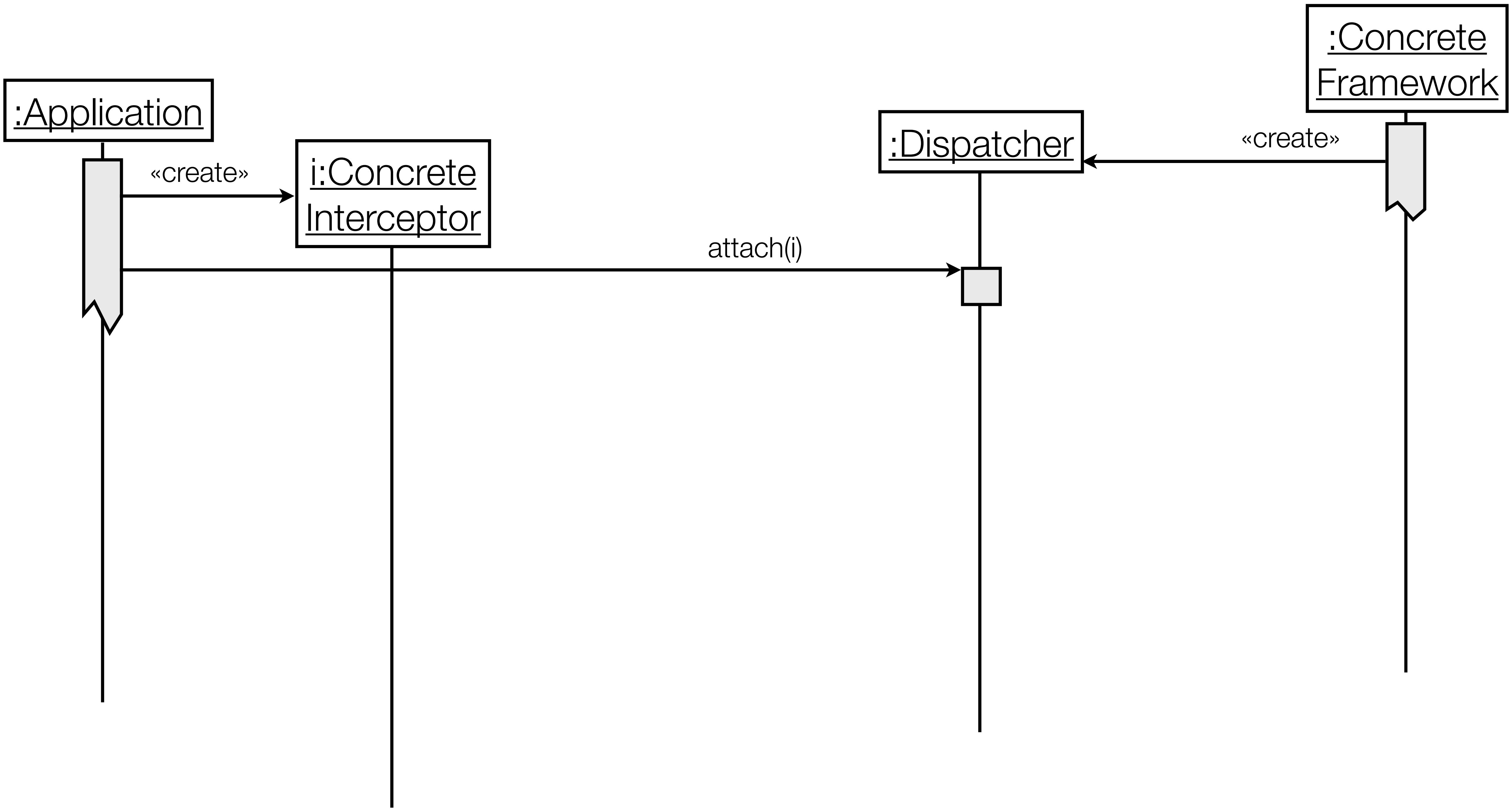
Responsibilities

- Allows services to obtain information from the concrete framework
- Allows services to control certain behavior of the concrete framework

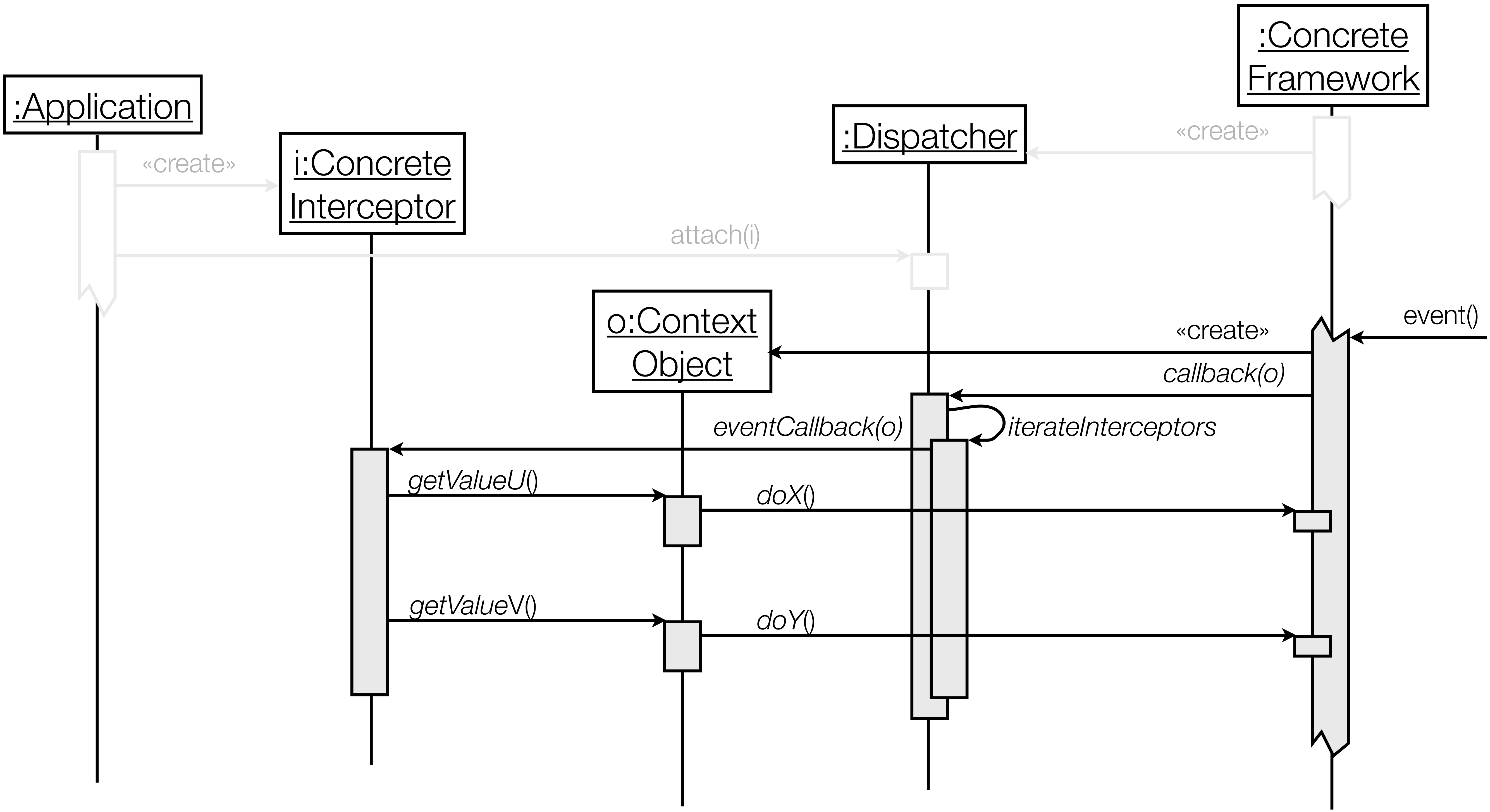
Identifying and Assigning Responsibilities

Class	Collaborations
Application	- Dispatcher - Concrete Interceptor
Responsibilities - Runs atop the concrete framework - Implements concrete interceptors and registers them with dispatchers	

Interaction (Initialization)



Interaction (Runtime)



Implementation Activities

Model the Internal Behavior of the Concrete Framework

- Model in particular those aspects related to interception
  (E.g. using state machines.)
  Example states in case of a framework for distributed applications:
  - initializing
  - marshaling request
  - demarshaling response
  - ...

Implementation Activities

Identify and Model Interception Points

- Identify concrete framework state transitions
- Partition interception points into reader and writer sets
  - *Reader Set*: the state transitions in which applications … information
  - *Writer Set*: the state transitions in which applications can modify the behavior of the concrete framework
- Integrate interception points into, e.g., the state machine model by introducing intermediary sets
- Partition interception points into disjoint interception groups (of semantically related interception points)
  For each group design a corresponding `Dispatcher` and `Interceptor` Interface

e.g., application shutdown...

Implementation Activities

Identify and Model Interception Points

Interception Point	Description	Reader / Writer
shutdown	The framework is shutting down. Clients can intercept this event to, e.g., free resources.	Reader
pre marshal out request	The (client) application sends a request to the remote object. Interceptors can be used to, e.g., encrypt the parameters.	Reader + Writer
...	...	...

Implementation Activities

Specify the Context Object

Recall, always ask yourself:
- what is necessary (you don't need it :-))
- what are my main architectural drivers
- ...

- Determine the context object's semantics
  - The information that is made available
  - How an interceptor is expected to control the framework's behavior (Forces: "extensibility" vs. "error proneness")
- Determine the number of context object types (E.g., `(Un)MarshaledRequest`)
- Define how to pass context objects
  - The `Context` object is passed to an interceptor when the interceptor is registered
  - The `Context` object is passed to a concrete interceptor with every callback invocation

Specify the Interceptors

Implementation Activities

The interceptor corresponds to the observer participant in the subject-observer pattern.

- For each interception point define a callback h
  Example:

```
public interface RequestInterceptor {

	void onPreMarshalRequest(UnmarshaledRequest context);

	void onPostMarshalRequest(MarshaledRequest context);
}
```

Implementation Activities

Specify the Dispatchers

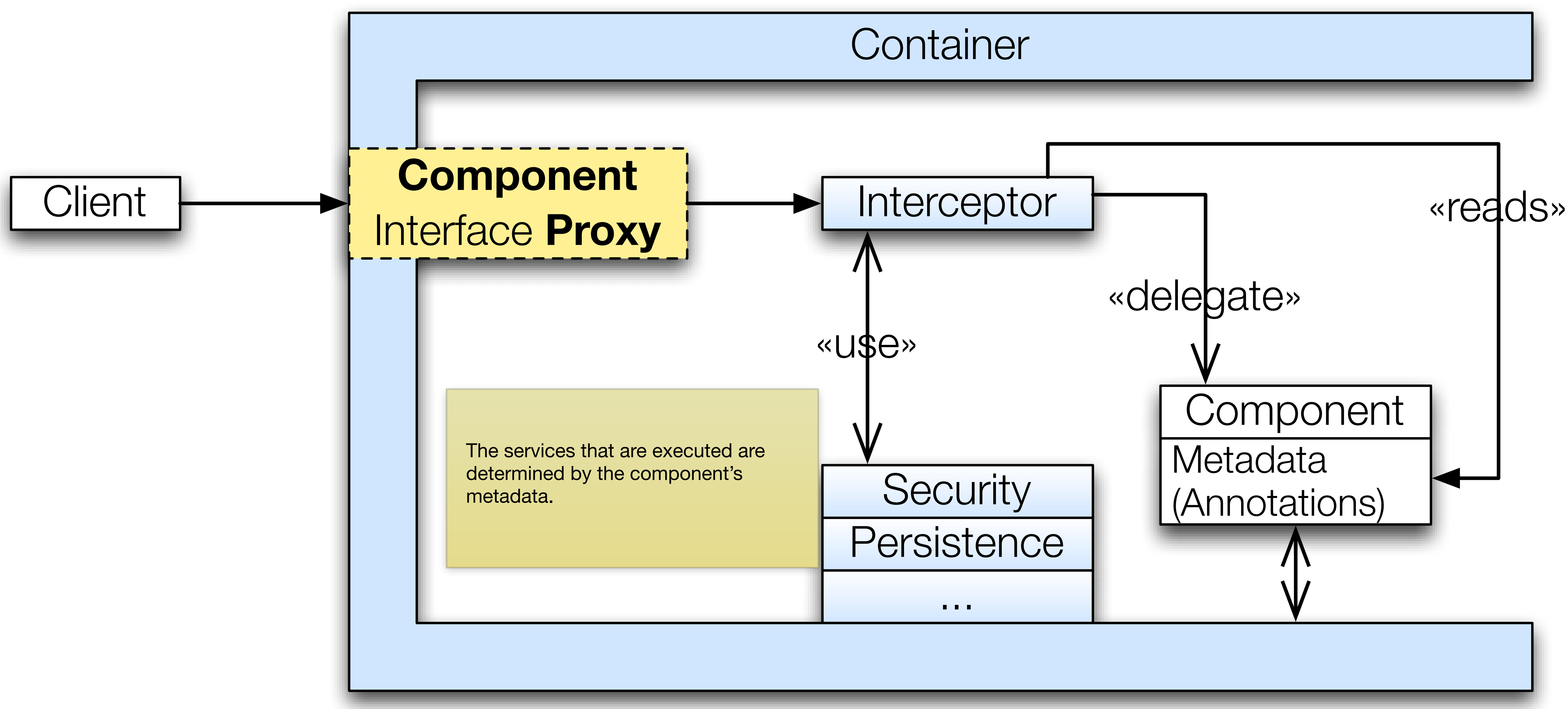
- Specify the interceptor registration interface
  (Application of the Subject-Observer Design Pattern)
- Specify the dispatcher callback interface
- *[If necessary:] To make the dispatching strategy exchangeable / adaptable apply the strategy pattern*

Implementation Variant - Interceptor proxy

- Often used on the server-side of a distributed system to intercept remote operations
- The concrete framework instantiates a proxy (Proxy Design Pattern) to an object residing on the server
  - The proxy implements the same interfaces as the object and intercepts all calls
  - The proxy performs the required service before forwarding the request to the local server object

Uses of the Interceptor Pattern

Component-based application servers...



[cf. p 133; Pattern-oriented Software Architecture Volume 2; D. Schmidt, M. Stal, H. Rohnert and F. Buschmann; Wiley 2000]]

Consequences

Interceptor proxy

- Extensible and flexible design
(Open-closed design principle)
- Separation of concerns; developers can focus on the application logic
- Interceptors can be reused across applications
- Complex design; number of different interceptors?
- Potential interception cascades if an interceptor changes the behavior of the concrete framework

Dr.-Ing. Michael Eichberg

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Components and Component-based Software Development

Introduction

Component-Based Software Development

an informal characterization

Component-based software development is the developing of software by **assembling pre-built (standard) components**.

Why components?

- Software is becoming increasingly large and complex
- Requirements are changing frequently; i.e. programs need to be adapted frequently
- Systematic reuse is required to deliver products on time
- Using "standard products" no competitive edge can be achieved
- Custom-made software is often too late

Hierarchies help to produce stable and flexible complexity.

Hierarchic systems are created much more rapidly from elementary constituents than non-hierarchic systems containing the same number of elements.

[Herber A. Simon; The Sciences of the Artificial; 3rd edition]

Possible components:
a component that provides authorization and authentication functionality
(often related to non-functional requirements)
a component that calculates the taxes for a product
(a domain specific component)
a (large-scale) component to render a webpage (e.g. the IE)
GUI widgets are sometimes also considered to be components
...

Idea / Goal of Component-Based Develop

also used: Component-Based Software Engineering (CBSE)

To provide support for the ***development of systems as assemblies of components****.*

To support the ***development of components as reusable entities****.*

To facilitate the maintenance and upgrading of systems by customizing and replacing their components

i.e. without recompiling / relinking the application (components)

[Building Reliable Component-Based Software Systems; Ivica Crnkovic and Magnus Larsson eds.; Artech House, 2002]

What is a component?

For a comprehensive overview of definitions related to the term “component”:

Component Software - Beyond Object-Oriented Programming, Second Edition;
Clemens Szyperski; Addison-Wesley 2002

What is a component?
(1st Definition)

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.*

[Workshop on Component-oriented Programming (ECOOP 96)]

What is a component?

(2nd Definition)

A software component is what is actually deployed - as an isolatable part of a system - in a component-based approach.

Characteristic properties of components:

- *is a **unit of independent deployment**;*
- *is a **unit of third-party composition**;*
- *has **no (externally) observable state**,*
  i.e. two copies of the same component have the same properties.

Separation of data (mutable instances) from the "plan"

[Clemens Szyperski; Component Software - Beyond Object-Oriented Programming; Second Edition; Addison-Wesley 2002]

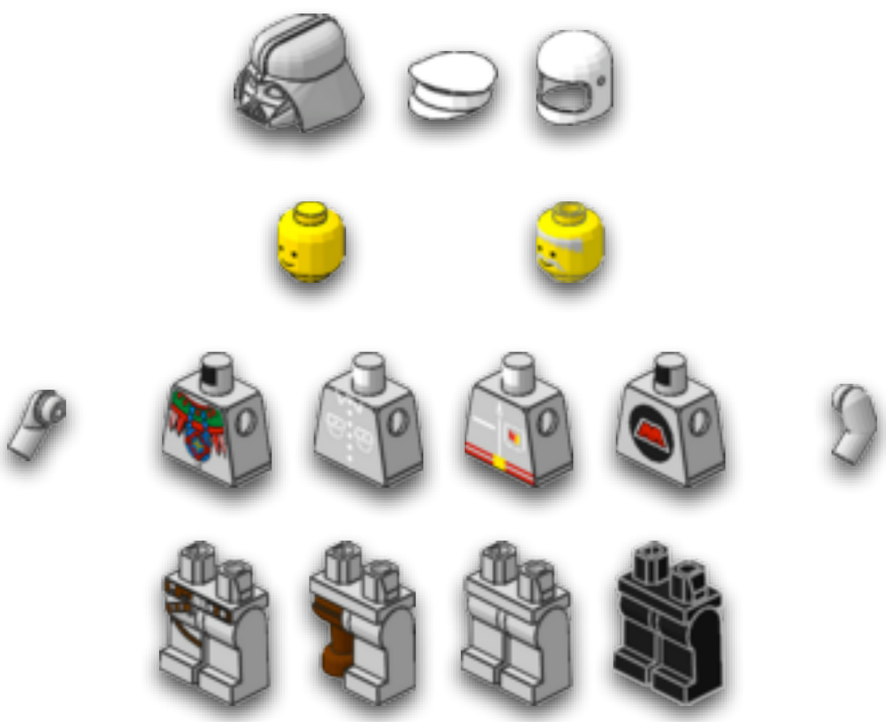
Software vs. Hardware Components

commonalities and differences

A lego “component” is a concrete instantiation of a blueprint - a software component is the blueprint.

Recall:
(e.g. from “Introduction to Software Engineering” or “Software Engineering Design and Construction”)

- Software is different from products in all other engineering disciplines
- Delivering software means **delivering the blueprint for products**
- Computers instantiate these blueprints; computers are factories
- A blueprint can be parameterized, instantiated multiple times,...



Lego “components” are concrete products

What is a component?

(3rd Definition - Software Component)

(Part I)

A software component is a ***software element that conforms to a component model****(…) and can be* ***independently deployed and composed*** *without modification* ***according to a composition standard****.*

[Heineman and Councill; Component-Based Software Engineering - Putting the Pieces Together; Addison-Wesley 2001]

What is a component?

(3rd Definition - Component Model)

(Part II)

A component model defines a ***specific interaction and composition standard****.*

A component model implementation is the dedicated set of executable software elements required to support the execution of components that conform to the model.

[Heineman and Councill; Component-Based Software Engineering - Putting the Pieces Together; Addison-Wesley 2001]

What is a component?
(4th Definition)

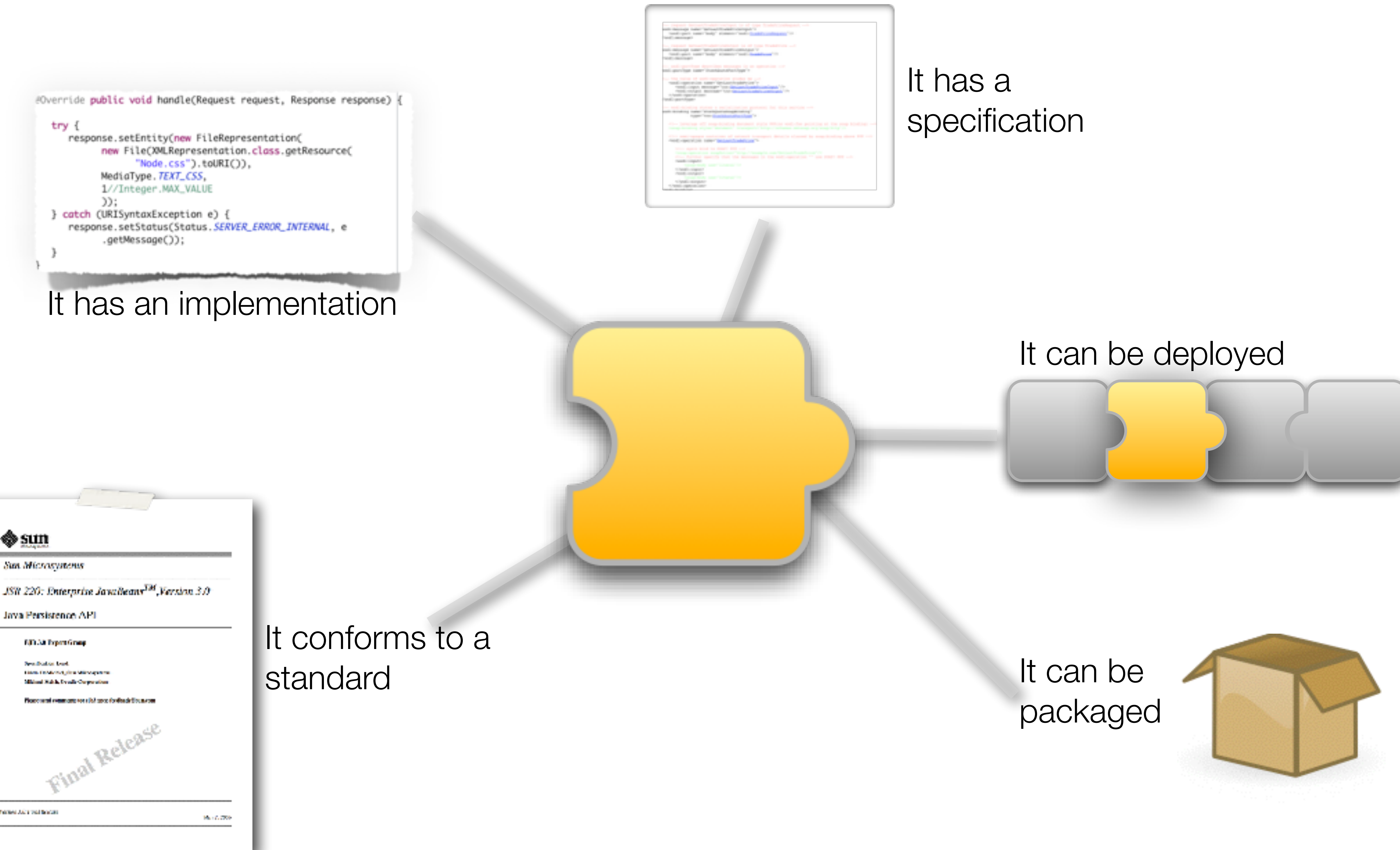
[...] While all these uses of the term component are valid [...] let's add additional properties to the definition:

- *A component is coarse-grained.*
- *They require a run-time environment.*
- *Remotely accessible.*

[...] this set of properties of components fits the so-called distributed, or server-side components[...].

[Völter, Schmid, Wolff; Server Component Patterns; Wiley 2002]

Elements of a Component



Elements of a Component

a specification

- Abstract description of services provided / required by the component
- A contract between provider and clients
- Usually more than the list of operations
  - Expected behavior of a component instance for specific situations
  - Constrains the allowable states of the component instance
  - Guide clients in appropriate interactions with the component instance (the order of interactions)
- In some cases formal, but most informal

Elements of a Component

an implementation

- One or more implementations
- Must conform to specification
- Specification allows a number of degrees of freedom on the internal operation of the component
- May be an existing system wrapped in such a way that its behavior conforms to the specification defined within the constraining component standard

```
@Override public void handle(Request request, Response response) {

   try {
      response.setEntity(new FileRepresentation(
            new File(XMLRepresentation.class.getResource(
                  "Node.css").toURI()),
            MediaType.TEXT_CSS,
            1//Integer.MAX_VALUE
            ));
   } catch (URISyntaxException e) {
      response.setStatus(Status.SERVER_ERROR_INTERNAL, e
            .getMessage());
   }
}
```

Often the case for Webservices that wrap legacy systems.

Elements of a Component

a packaging approach

- Components can be grouped in different ways to provide a replaceable set of services
- Typically these are packages that are bought and sold when acquiring components from third parties
- Each package provides a unit of functionality to be installed in the system
- Some sort of registration of the package within the component model is expected (registry)

Elements of a Component

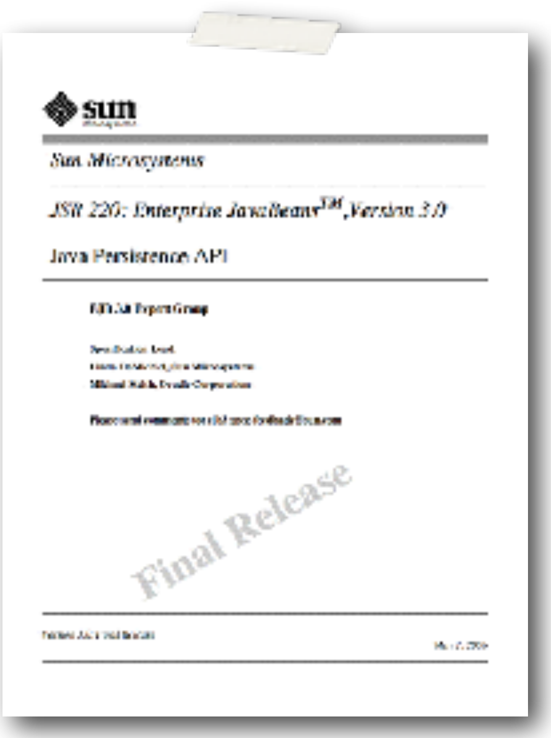
a deployment approach

- Once installed a packaged component will be deployed
- Delopyment means creating an executable instance of a component and allowing interactions with it to occur
- A component may be deployed multiple times; each instance is unique

Elements of a Component

a standard

- A set of **standard services that can be assumed by components and assemblers of component-based systems**
  - E.g., directory services, security, transaction management, scripting, etc.
  - The services are provided to components in a transparent way
    Components do not need to explicitly call the services.
- A set of rules that must be obeyed by the component in order for it to take advantage of the services transparently



Developing Components

A component has to have a […] large number of uses […] for it to be viable.

As a rule of thumb, most components need to be used three times before breaking even.

[…] two separate, from-scratch development efforts are still cheaper than a single effort to produce a more generic component.

[p. 12, Clemens Szyperski; Component Software - Beyond Object-Oriented Programming, Addison-Wesley 2002]

Summary

Hence, an interface standard is required that declares what can comprise an interface.

Components have to have...

- ...**clearly defined interfaces**:
  - components support a **provided interface**
  - a component needs a **required interface** if the component requests an interaction defined in that interface
- ...**an interaction standard** that covers all interactions that may exist between components; it specifies the explicit context dependency a component may have
- ...**a component model** that defines:
  - how to construct a component,
  - how to deploy a component,
  - how components have to interact (the interaction standard)

Another component has to support that interface.

Dr.-Ing. Michael Eichberg

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Software Component Infrastructures

Introduction

Purpose of Software Component Infrastructures

The component implements the functional concerns while the component infrastructure provides the non-functional concerns.

*The purpose of a component infrastructure is to **separate responsibilities** and to ensure that **logical connections between components do not result in unnecessary coupling**.*

[Steve Latchem, Component Infrastructures: Placing Software Components in Context; in Component-Based Software Engineering, Addison Wesley 2001]

Properties of Software Component Infrastructures

A software component infrastructure should possess the following properties or enable components to provide:

- *location transparency - **a component should be useable independently of its location** (within the same process, another process, a different computer,…).*

This is not strictly required.

[Componentware, Frank Griffel,dpunkt.verlag 1998]

Properties of Software Component Infrastructures

A software component infrastructure should possess the following properties or enable components to provide:

- ***strict separation of interface and implementation*.**

In EJB it was a best practice not to implement the interface directly!
(This, however, violates common best practices and - in particular - renders refactoring tools useless!)

[Componentware, Frank Griffel,dpunkt.verlag 1998]

Properties of Software Component Infrastructures

A software component infrastructure should possess the following properties or enable components to provide:

- ***a self-describing interface** - to enable a better reuse and to enable runtime discovery of a component, a component should provide extensive information about the provided functionality and how it can be accessed.*

[Componentware, Frank Griffel,dpunkt.verlag 1998]

Properties of Software Component Infrastructures

A software component infrastructure should possess the following properties or enable components to be:

- ***composable*** *- i.e. components should be composable and integrable to form new components.*

[Componentware, Frank Griffel,dpunkt.verlag 1998]

Dr.-Ing. Michael Eichberg

TECHNISCHE UNIVERSITÄT DARMSTADT

Designing Software Component Infrastructures

Designing Software Component Infrastructures

*When designing the software component infrastructure, **you must have a base set of applications in mind**.*

e.g., Enterprise Applications or Smart Homes,...

[Len Bass, Software Architecture Design Principles; in Component-Based Software Engineering, Addison Wesley 2001]

Designing Software Component Infrastructures

During design the architectural drivers have to be identified*, e.g., maintainability and extendibility, performance, throughput, reuse...*

[Len Bass, Software Architecture Design Principles; in Component-Based Software Engineering, Addison Wesley 2001]

Designing Software Component Infrastructures

*Only the externally visible functions and **behavior of the components become part** of the design of the software component infrastructure.*

E.g., if we want to have components that represent sessions, we have to identify / specify the expected behavior of all these components and to develop functionality to support such components.

[Len Bass, Software Architecture Design Principles; in Component-Based Software Engineering, Addison Wesley 2001]

Designing Software Component Infrastructures

*The software **component infrastructure embodies the fundamental tradeoffs** and decisions made during design, which are recorded as design rules.*

Quality Attribute	Architectural Mechanism
Modifiability	Separation, Indirection
Reliability	Redundancy
...	...

[Len Bass, Software Architecture Design Principles; in Component-Based Software Engineering, Addison Wesley 2001]

Related class design principles: Open-closed principle...

Designing Software Component Infrastructures

e.g., security, logging, transactions, passivation, pooling,...

To determine the shared services to be implemented within the software component infrastructure, the base set of applications must be at least partially designed; ***the software component infrastructure can not be designed in isolation****.*

[Len Bass, Software Architecture Design Principles; in Component-Based Software Engineering, Addison Wesley 2001]

Designing Software Component Infrastructures

[…] it is important to design component infrastructures following the principle of ***separating concerns****.*

Separating concerns between the infrastructure and the components that will use it!

[Steve Latchem, Component Infrastructures: Placing Software Components in Context; in Component-Based Software Engineering, Addison Wesley 2001]

Designing Software Component Infrastructures

[In case of CBSD:] Software architecture [is used to] refer to a specific software component infrastructure ***with an associated set of design rules****.*

[Len Bass, Software Architecture Design Principles; in Component-Based Software Engineering, Addison Wesley 2001]

Designing Software Component Infrastructures

rules & implementation restrictions

- Rules are conventions about naming methods, how to specify metadata, etc.
- In addition, the container has to make certain assumptions about the behavior of components in order to control their life cycle
  E.g., how to manage synchronization if components would create threads.
- Hence, the container imposes implementation restrictions on component implementations
  E.g., threading restrictions, presence of GUI, assumptions about locality, etc.

Designing Software Component Infrastructures

rules & implementation restrictions

- many **restrictions** which components have to follow **cannot be enforced**, neither statically nor dynamically
- *Saying: "…if you follow the rules of the component framework and obey its implementation restrictions, your components will be transactional, secure, …, and will run in all servers that implement the architecture …"*

!

Dr.-Ing. Michael Eichberg

TECHNISCHE UNIVERSITÄT DARMSTADT

Basic Building Blocks of Software Component Infrastructures

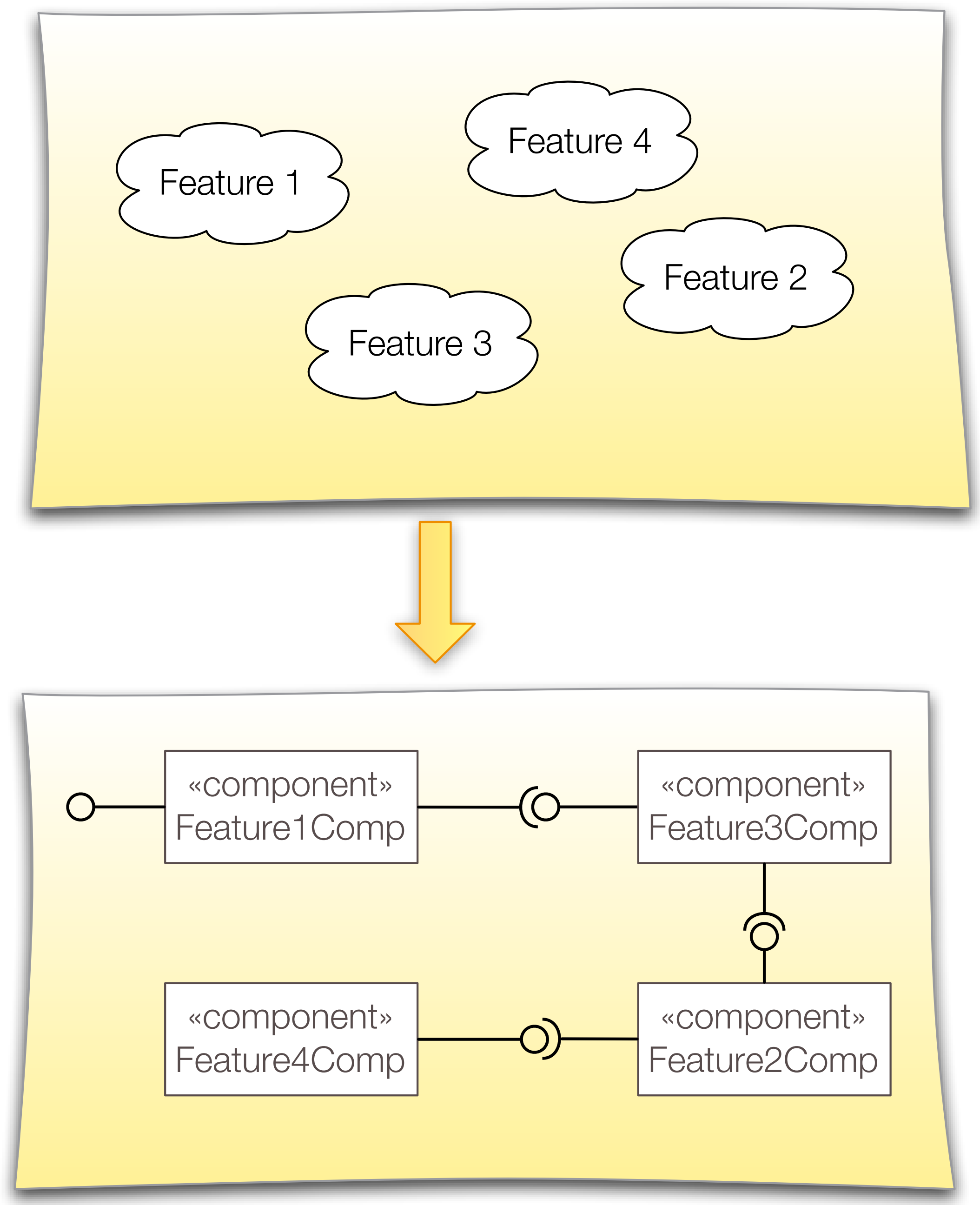
Server Component Patterns;
Markus Völter, Alexander Schmid, Eberhard Wolff;
Wiley 2002

Component

A component should NEVER depend on the internals of another component.

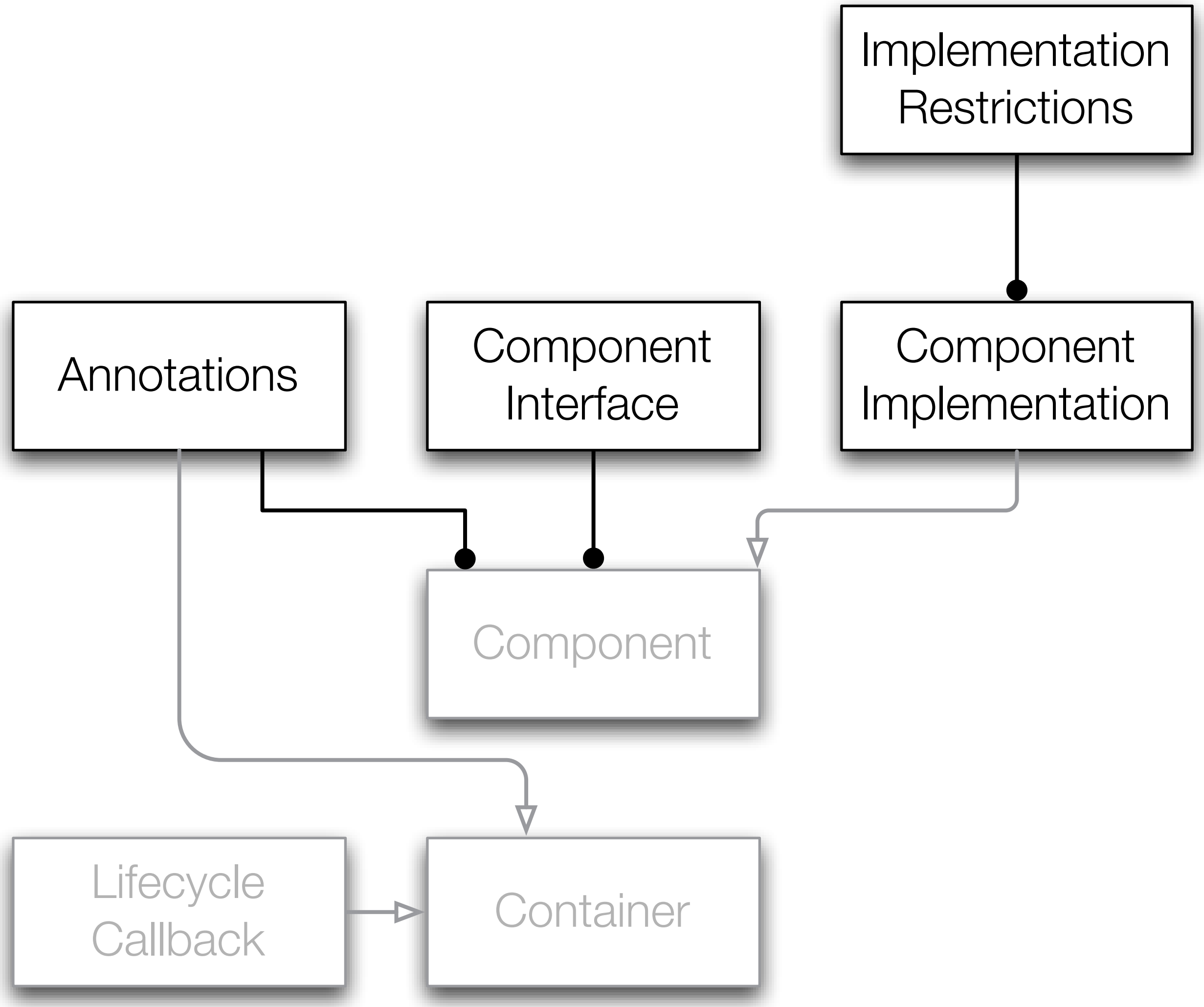
- Decompose an application's **functionality in distinct components**
- A component is **responsible for providing one part** of the overall functionality
- A component implements its responsibilities without introducing strong dependencies
- **A component should exhibit:**
  - **high cohesion, and**
  - **loose coupling**

other way round: components (typically) implement the functional requirements



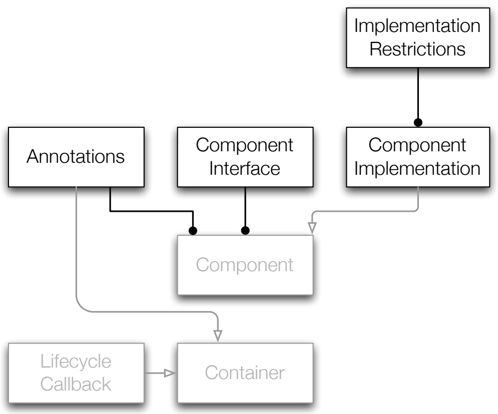
Building Blocks that Make Up Components

- Clients access components through the **component's interface**
- The **component implementation** is the *implementation of the functional requirements*
- **"Annotations"** are used to tell the container which technical concerns should be added to them

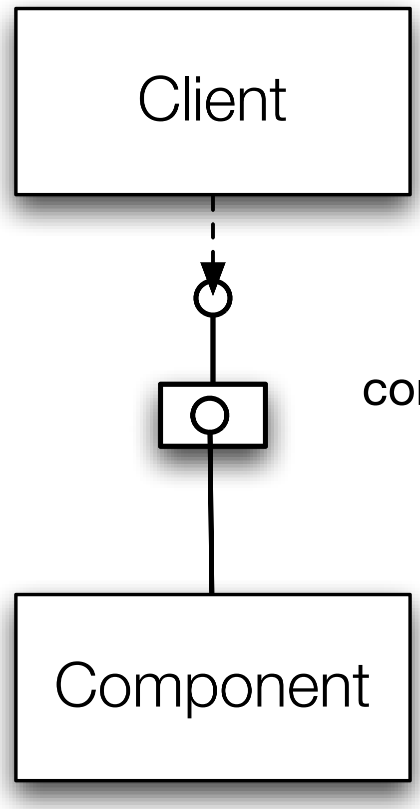


this line type (A •- B) means: "A provides context for B" (i.e. we can have A without B, but not the other way round.)

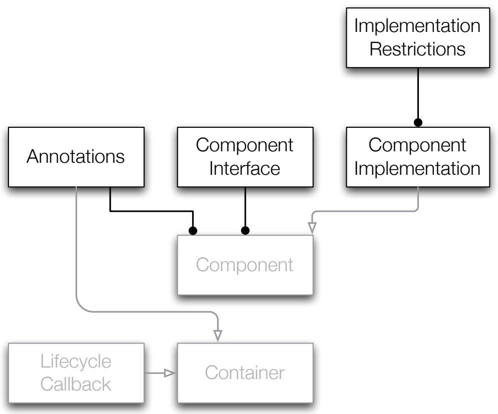
Component Interface



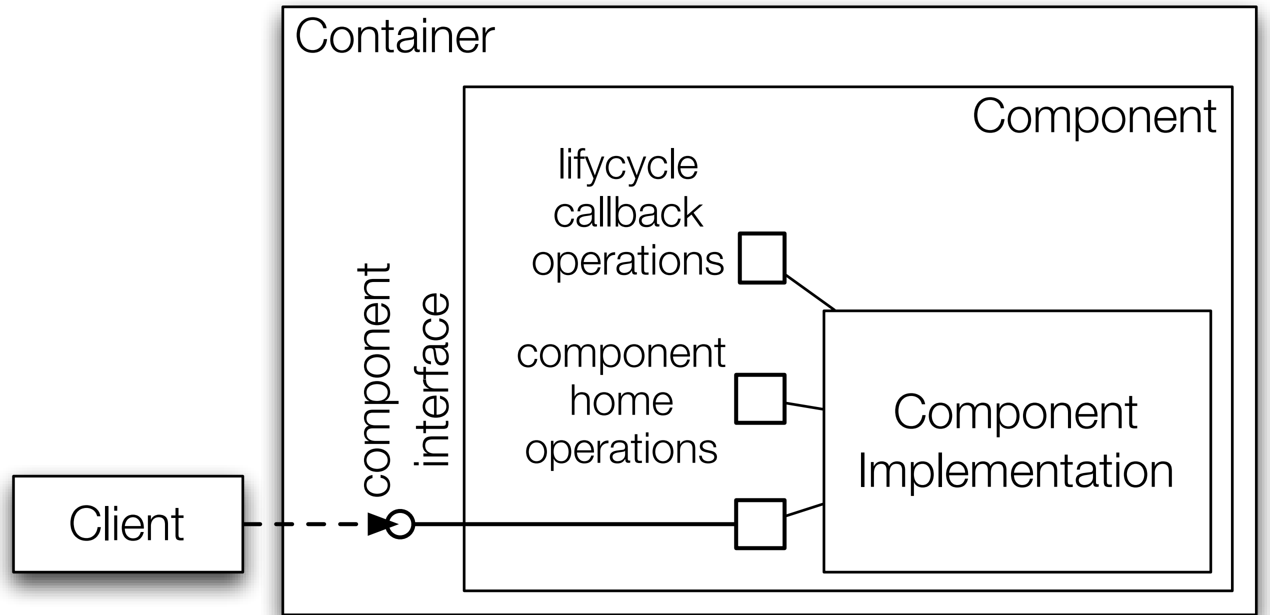
- The **interface defines what** a component does and **not how**; it serves as a **contract between client and component**
  - The operations provided by the component and their signatures
  - Ideally, it defines the semantics of a component
- Using the interface it is possible to decouple components (implementations)
- An explicit component interface makes it possible to have multiple implementations; to **evolve components independently** of each other



Component Implementation



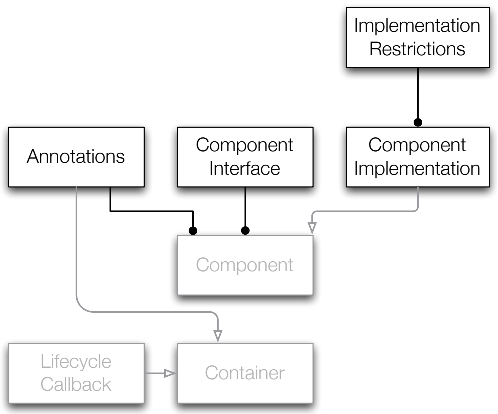
- Provides **operations to instantiate a component** (Sometimes defined by a - *so called* - component home interface.)
- Implements lifecycle callback operations
- The component implementation should be separate from the component interface
  It is the job of the container to attach a component interface to the componen



This is not always strictly required... it depends on the technical concerns that are / may be provided by the container.

Component Home Operations

Example (J2EE 1.4 - EJB Home Interface for Session Beans)



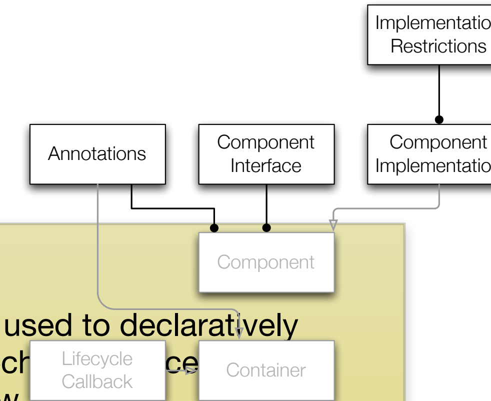
- [For a session bean,] the purpose of the home interface is to define the create methods [that a remote client can invoke].
  (The container creates the component instances.)

```
Cart shoppingCart = home.create("Duke DeEarl", "123"); // usage scenario

import javax.ejb.EJBHome;
public interface CartHome extends EJBHome {
  Cart create(String person) throws RemoteException, CreateException;
  Cart create(String person, String id) throws RemoteException, CreateException;
}

// Bean Implementation
 public void ejbCreate(String person, String id) throws CreateException {
   // initialize bean
 }
```

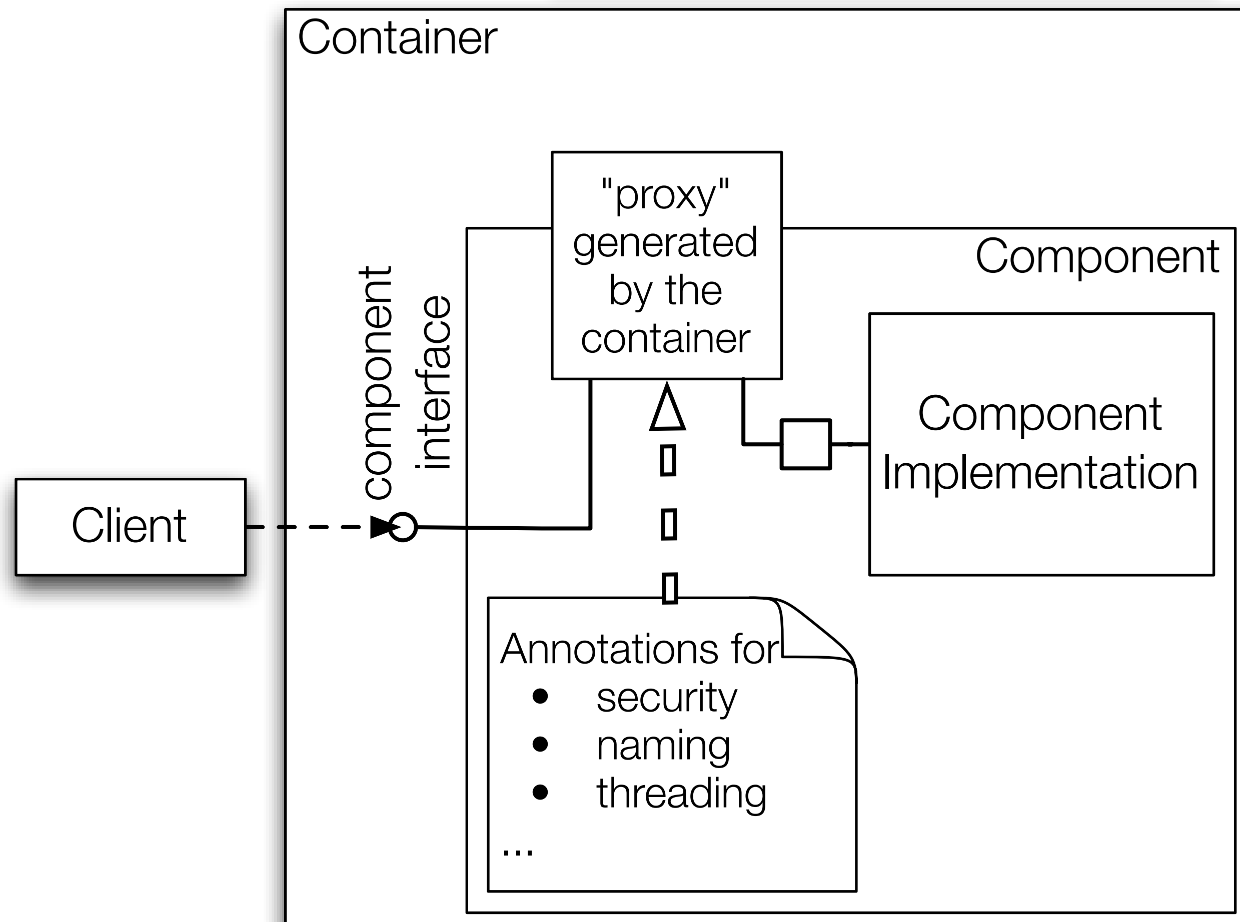
Annotations

The container cannot guess which non-functional concerns are required.

Annotations are used to declaratively specify which tech ce required and how.



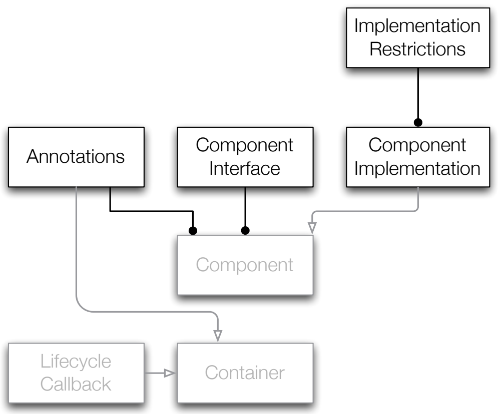
- Annotations are used to configure the technical concerns that are required
  The container (the component's runtime environment) provides the implementation.
- Annotations are used to configure the container, e.g.
  - transaction handling
  - security
  - …



Specifications of the configurations for the non-functional concerns should not pollute the component's implementations.

Annotations

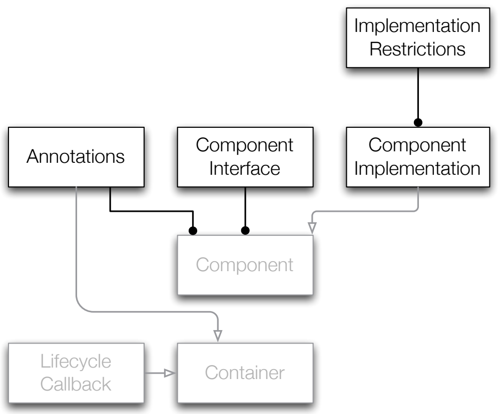
Example



- The **`TransactionBean`** class's transaction attribute (part of Java EE > 5) is **`NotSupported`**, firstMethod's transaction attribute is **`RequiresNew`**, and secondMethod's attribute is **`Required`**.
- A method-level attribute overrides a class-level attribute

```
@TransactionAttribute(NOT_SUPPORTED) @Stateful
public class TransactionBean implements Transaction {

    @TransactionAttribute(REQUIRES_NEW)
    public void firstMethod() {...}

    @TransactionAttribute(REQUIRED)
    public void secondMethod() {...}

    public void thirdMethod() {...}
    public void fourthMethod() {...}
}
```

Implementation Restrictions



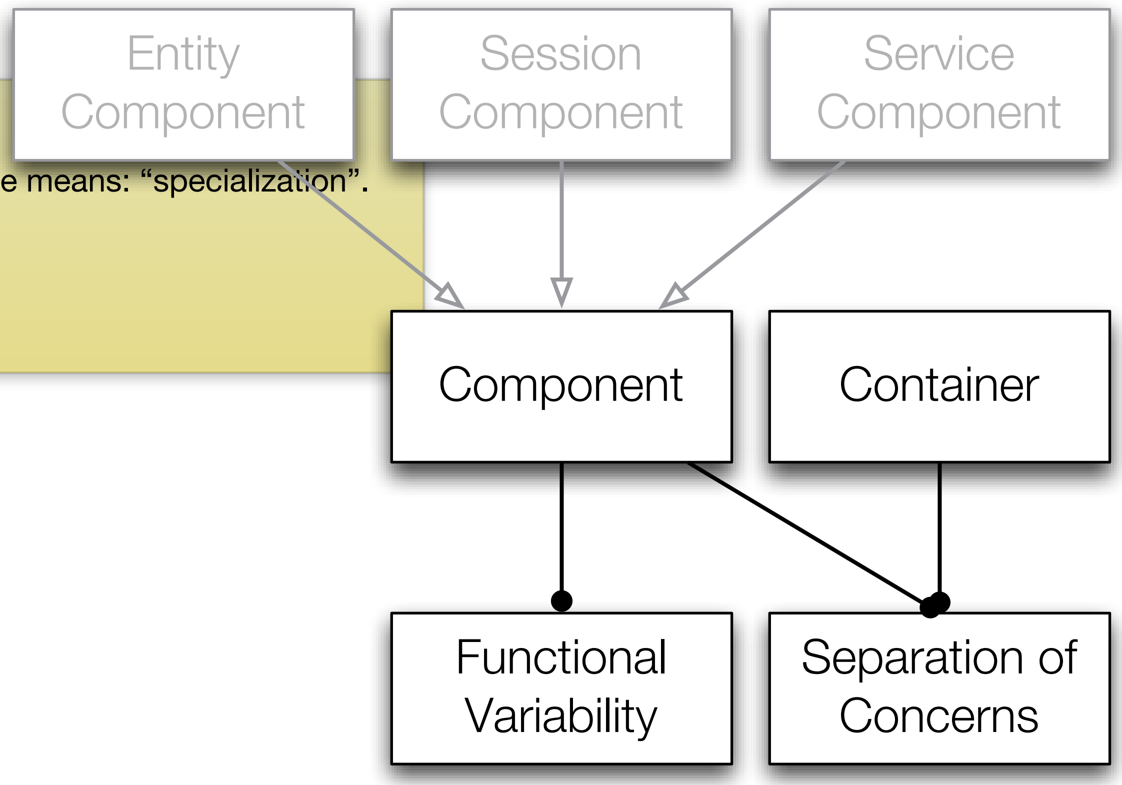
- The runtime environment (has to) makes certain ***assumptions about the behavior*** of the components
- These assumptions result in implementation restrictions that components have to follow
- The specific implementation restrictions vary widely and can be related to the **use of specific APIs or programming language features**

Core Infrastructure Elements

fundamental building blocks

Component types typically found when developing distributed enterprise applications.

this line type means: “specialization”.



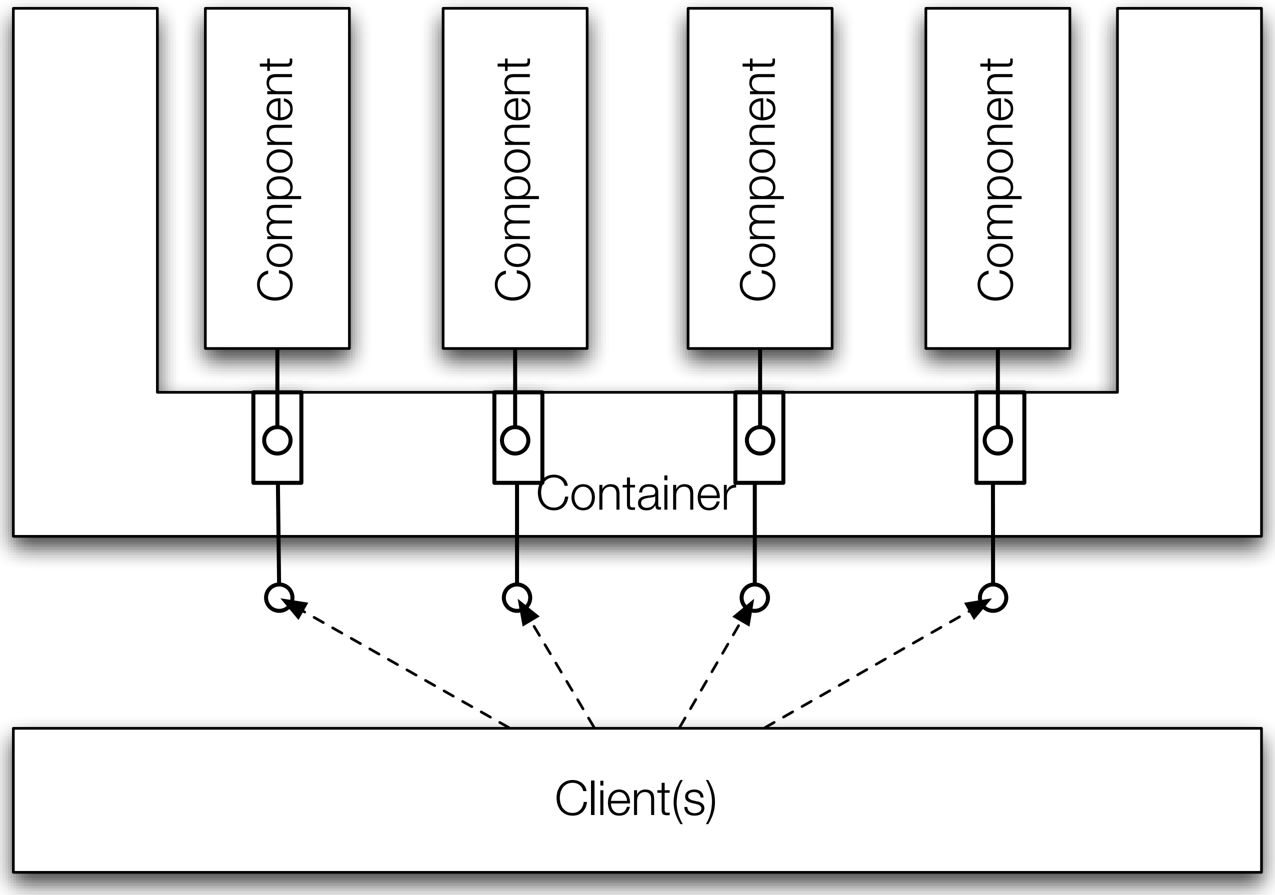
- A component implements some well-defined functionality
- The **container provides a run-time environment for components**, adding the technical concerns

this line type (A •- B) means: “A provides context for B” (i.e. we can have A without B, but not the other way round.)

Core Infrastructure Elements

Container

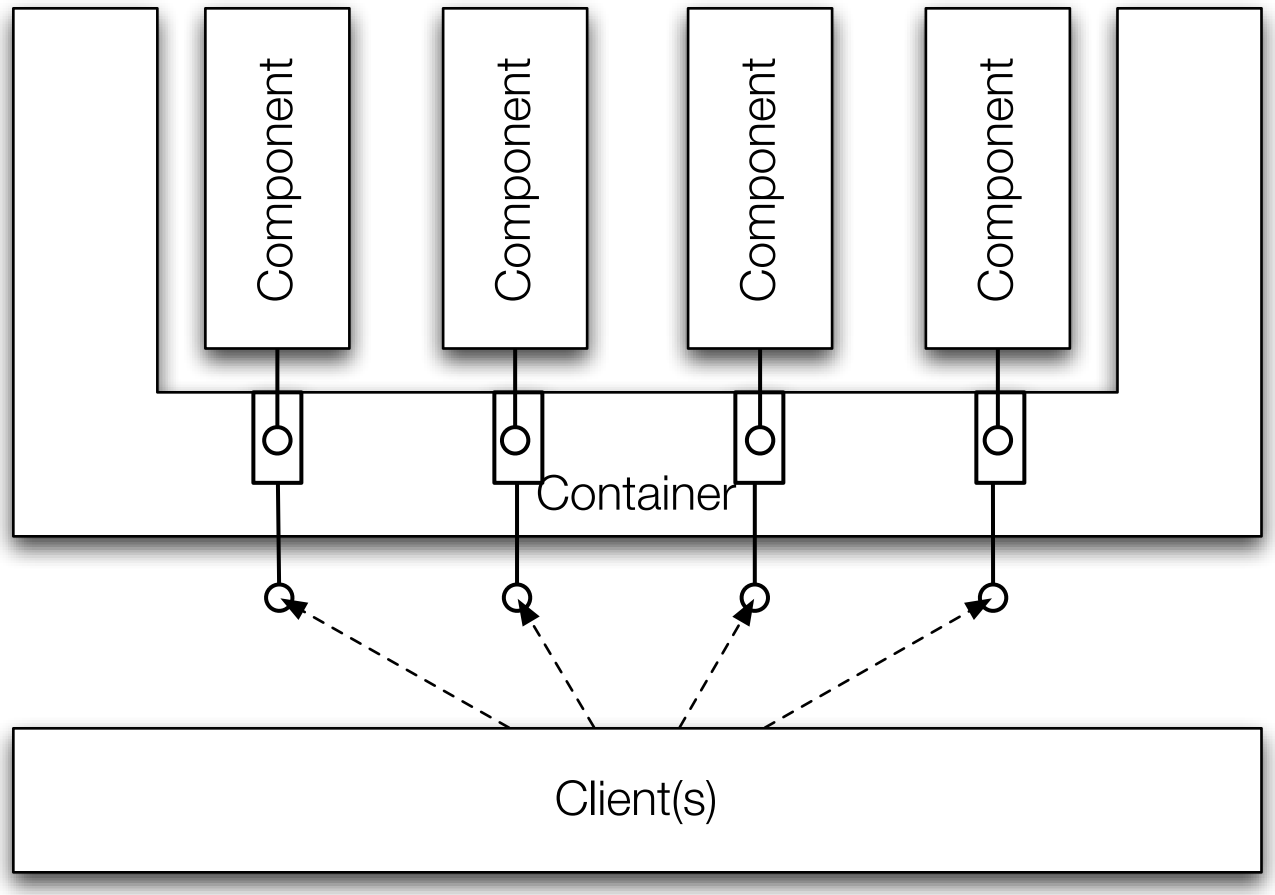
- The **container provides the technical concerns** and integrates the components
- Conceptually, a container wraps the components, giving *clients the illusion of tightly-integrated functional and non-functional concerns*



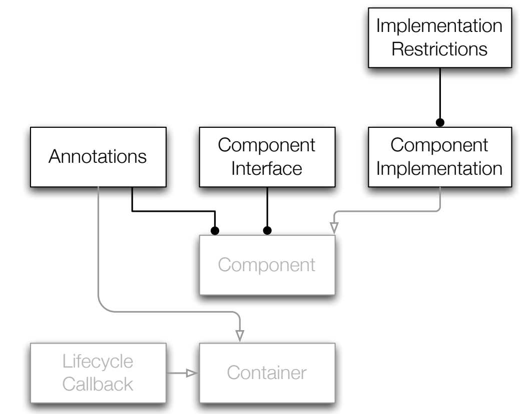
Core Infrastructure Elements

Container

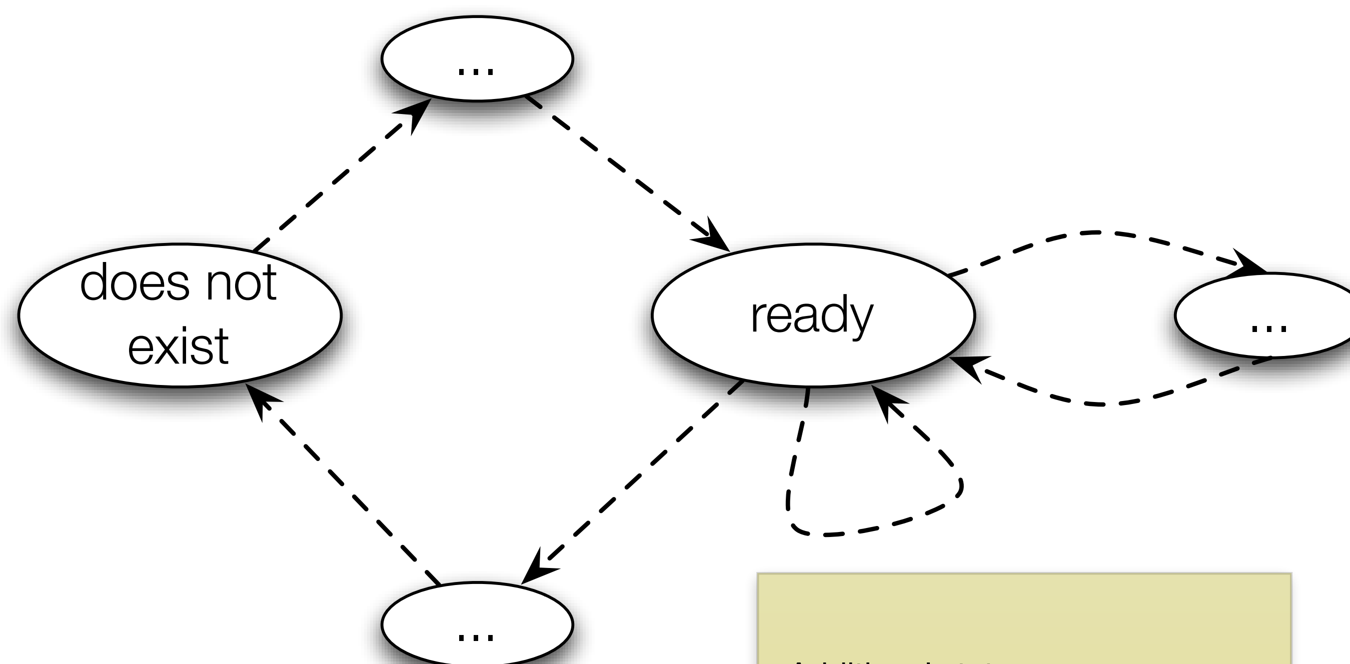
- Typically, **one container exists for each component type**
- The **container controls the lifecycle** of the component instances



Lifecycle Callback Operations



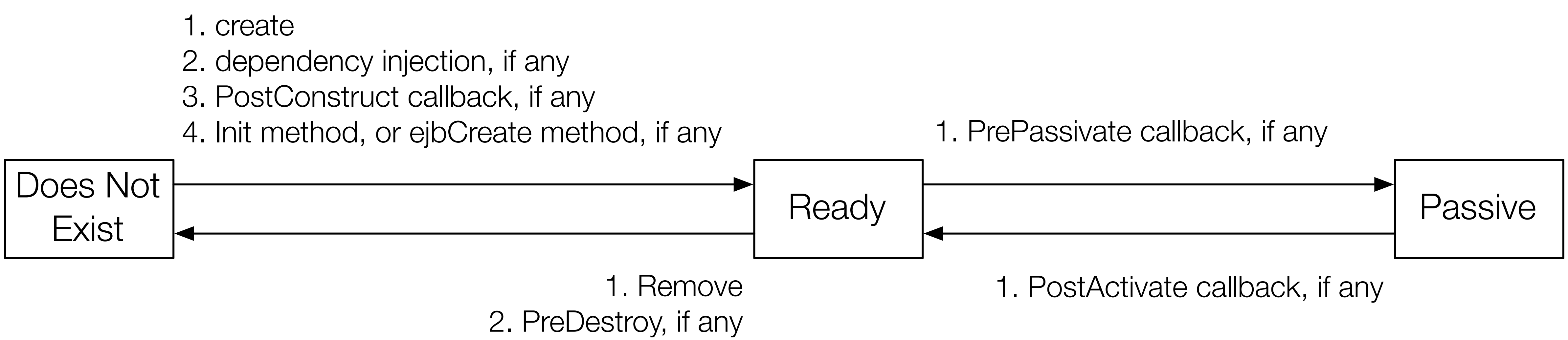
- Components - at least - need to be initialized and destroyed (lifecycle management)
- Lifecycle operations are often **responsible to acquire and release resources**
- The lifecycle callback operations are called at well-defined points during the life-cycle
- The lifecycle operations depend on the type of components



Additional states:
- pre-init
- pre-destroy
- before passivation
- passivated
- after passivation
- pooled
- ...

Lifecycle Callback Operations

Example



The Life Cycle of a Java EE 5 Stateful Session Bean

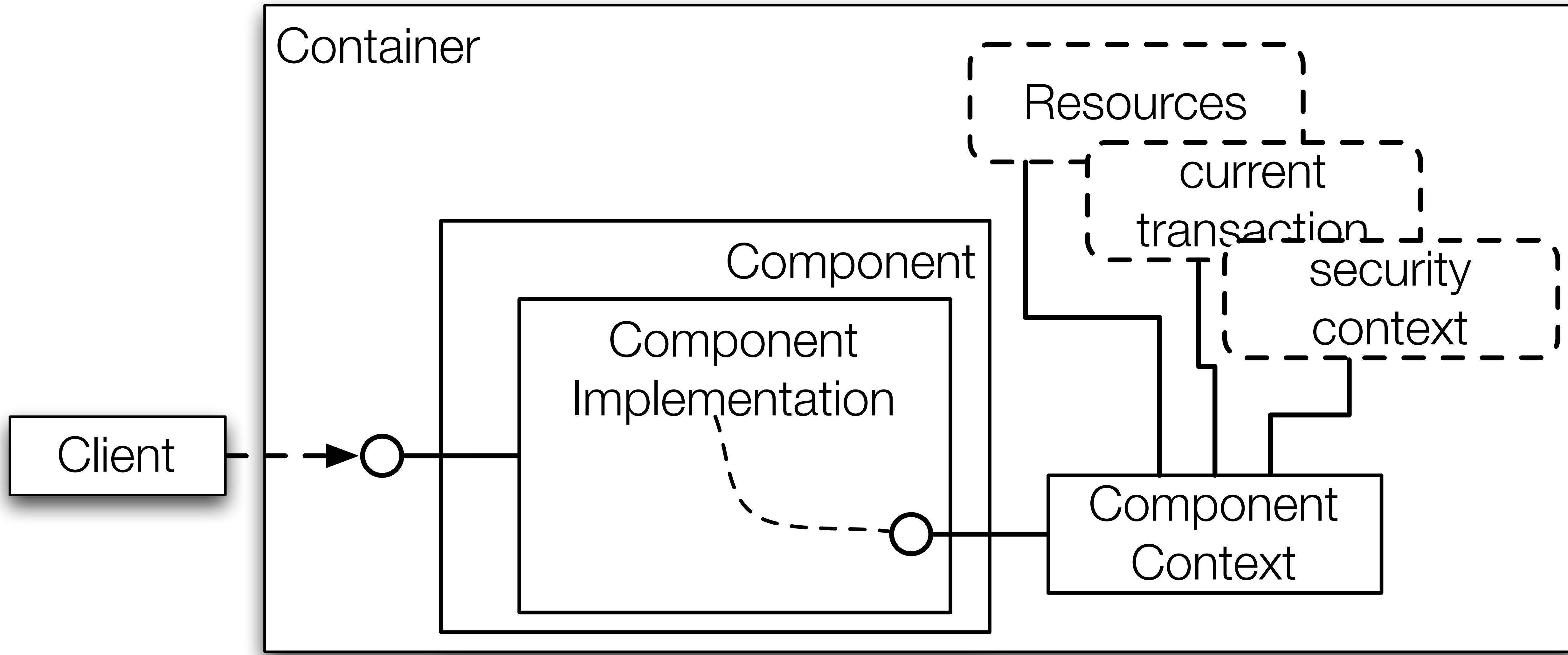
http://java.sun.com/javaee/5/docs/tutorial/doc/bnbmt.html#bnbmu

A Component and its Environment

- Components cannot exist completely on their own - they have to be given access to external resources
- Generally, components are not responsible for the implementation of technical concerns, but they might need to control some aspects of them at run time (without compromising the integrity of the CONTAINER.)

A Component and its Environment

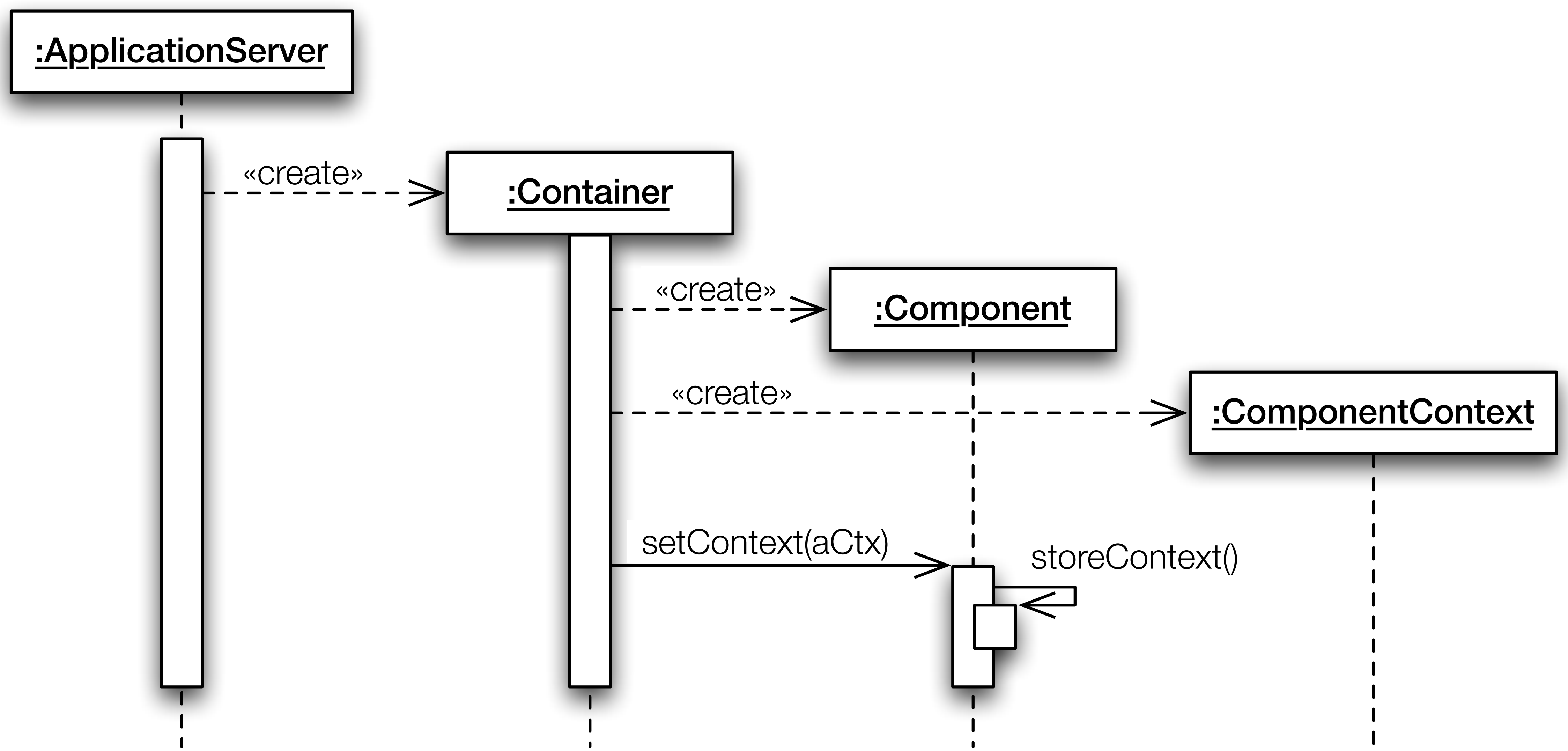
Component Context



The Container provides a Component Context to each Component Instance. This context object's interface provides operations for accessing resources, security information,…. It can also include the possibility of accessing other parts of the component's environment.

A Component and its Environment

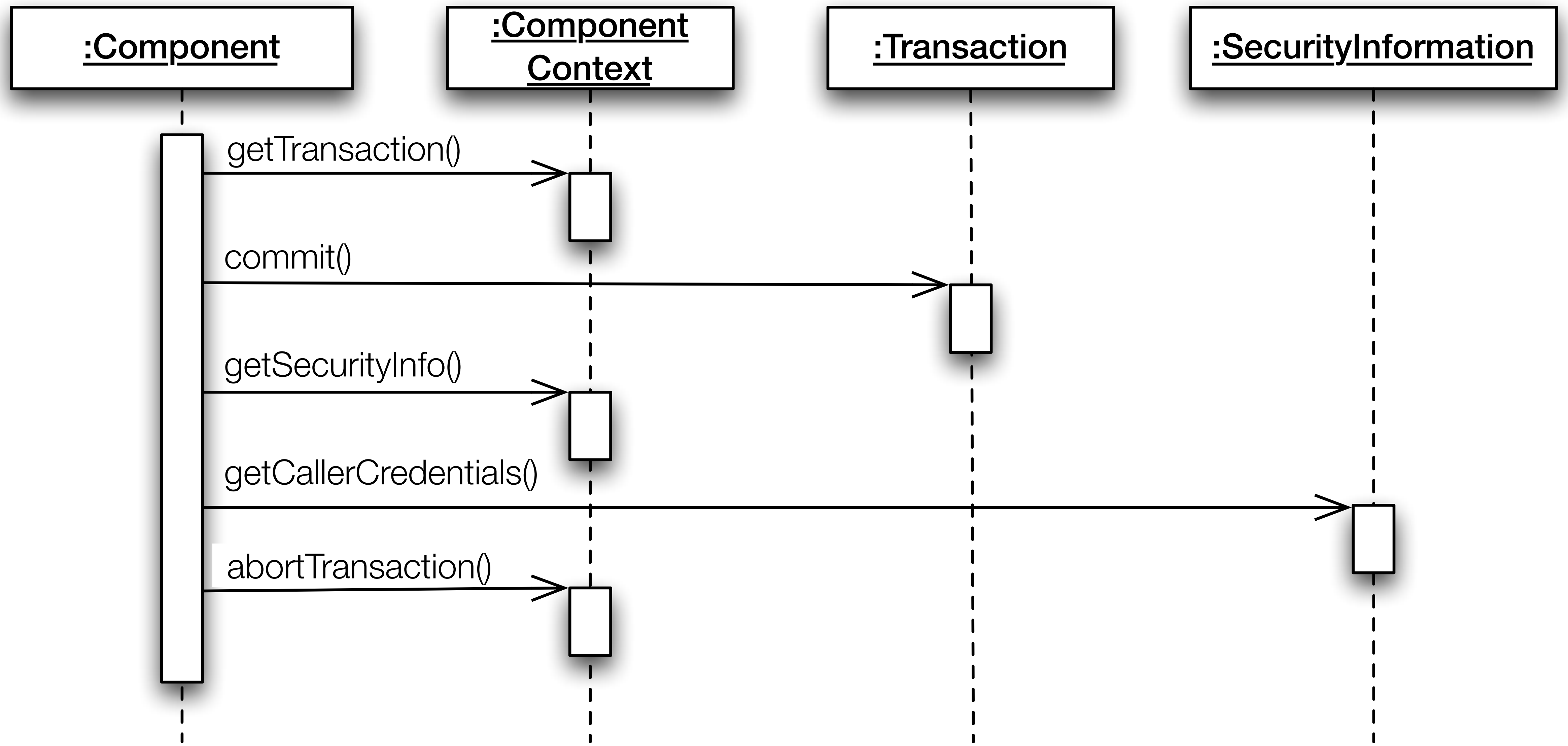
Component Context



Initialization

A Component and its Environment

Component Context



Runtime

Dr.-Ing. Michael Eichberg
eichberg@informatik.tu-darmstadt.de
TECHNISCHE UNIVERSITÄT DARMSTADT

OSGi

OSGi

About OSGi

The OSGi specifications define a **standardized**, **component oriented**, computing **environment** for **networked services** that is the foundation of an enhanced **service oriented architecture**.

OSGi

The OSGi Service Platform is a Java based application server for networked devices…

[Technical Whitepaper, Rev. 4.1, OSGi Alliance, November 2005]

OSGi

The business perspective:

The OSGi Service Platform is […] considered to be the cheapest, fastest and easiest way to enable the ***dynamic deployment of Web 2.0 services*** *and mashups in the next generation Java Service Platform.*

[from the OSGi Website (April 2007)]

OSGi

The technical perspective:

The OSGi specifications [...] form ***a small layer*** *that allows multiple, Java based,* ***components to efficiently cooperate*** *in* ***a single Java Virtual Machine****.*

[About the OSGi Service Platform; OSGi Alliance, November 2005]

Scope of the OSGi Specifications

- A standard […] software component framework for manufactures, service providers, and developers.
- A model for co-existence of different components / applications in a single JVM...
- A cooperative model where applications can dynamically discover and use services provided by other applications *running inside the same OSGI Service Platform*.

Scope of the OSGi Specifications

- A […] deployment Application Programming Interface (API) that controls the life-cycle of applications.
- A **secure environment** that **executes applications in a sandbox** so that these applications cannot harm the environment, nor interfere with other resident applications.
- A number of standardized, optional services: Logging, Configuration, […]

OSGi

Definition of “Component”

Software **components** are libraries or applications that can **dynamically discover and use other components**.

i.e. each component has to define which other services (components) the component requires and / or provides!

OSGi

Design of the Platform

Main drivers for the design of the OSGi Service Platform:

- 24/7 operation
- deployable on embedded systems

Resulting design decisions:

- dynamism; i.e. starting, stopping and updating / replacing services must be possible at runtime
- memory consumption has to be minimized; running multiple applications in a single JVM must be supported

OSGi - an Overview

Several implementations of the standard exist; available “Service Platforms”:

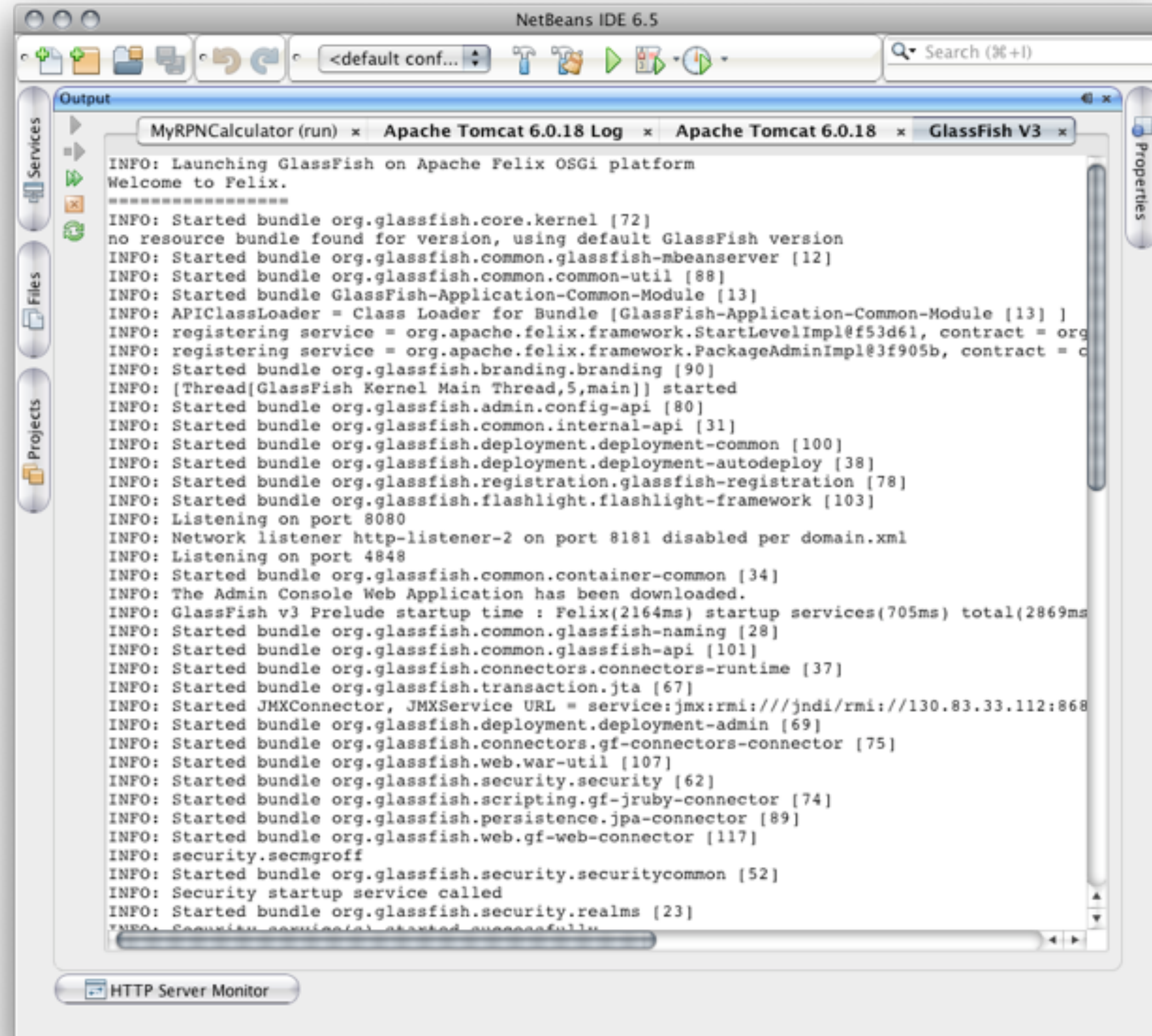
- Gatespace Telematics Knopflerfish
  www.knopflerfish.org
- ProSyst Software mBedded Server
  www.prosyst.com
- Eclipse Equinox
  www.eclipse.org/equinox/
- Apache Felix
  felix.apache.org

Wide adoption of the OSGi specification and OSGi specification based products:

- Nokia
- Siemens
- BMW
- Volvo
- Cisco
- Wind River
- Bombardier
- ...

e.g. BMW uses the OSGi specifications as the base

Glassfish v3 uses Apache Felix OSGi



OSGi

Standard Components and Services

- Log Service
- Http Service
- Device Access
- Preferences Service
- User Admin Service
- Wire Admin Service
- XML Parser Service
- Event Admin Service
- ...

In general, these are only basic services!

Functionality of the OSGi Framework

The **framework** is the core of the OSGi Service Platform Specifications and provides a ...

- general-purpose,
- secure and
- managed

Java framework that **supports the deployment of extensible applications** (bundles).

Functionality of the OSGi Framework

Building Blocks/“Layers” of the Framework

- ***Security*** *Layer*
  ...provides the infrastructure to deploy and manage applications that must run in controlled environments.
- ***Module*** *Layer*
  ...supports packaging, deploying, and validating Java-based applications and components
- ***Life Cycle*** *Layer*
  ...provides an API to control the security and life cycle operations of bundles.
- ***Service*** *Layer*
  ...defines a dynamic collaborative model. The service model is a publish, find and bind model.
- Actual Services

Functionality of the OSGi Framework

Layers of the Framework (main focus of this lecture)

- Security Layer
  ...provides the infrastructure to deploy and manage applications that must run in controlled environments.
- Module Layer
  ...supports packaging, deploying, and validating Java-based applications and components
- Life Cycle Layer
  ...provides an API to control the security and life cycle operations of bundles.
- Service Layer
  ...defines a dynamic collaborative model. The service model is a publish, find and bind model.
- Actual Services

Security Layer

(Further details can be found in the OSGi core specification.)

- The layer is **based on the Java 2 security architecture** and targets **code authentication**:
  - by location
  - by signer
- This information is used to grant permissions based on the authenticated principal or to restrict the set of bundles that can be managed by another bundle
- Signing is based on Java2 JAR signing and uses public key cryptography
- The **security layer is optional**. (i.e. it is possible to implement the interfaces using stubs and to grant all bundles all permissions)

Module Layer: Bundles

(Details regarding native code loading can be found in the OSGi core specification.)

- The **unit of modularization** is called a bundle
- A bundle **is comprised of all resources**, that together can provide functions to end users
- Bundles **can share Java packages** among an exporter bundle and an importer bundle in a well-defined way
- Bundles are the **only entities for deploying Java-based applications**

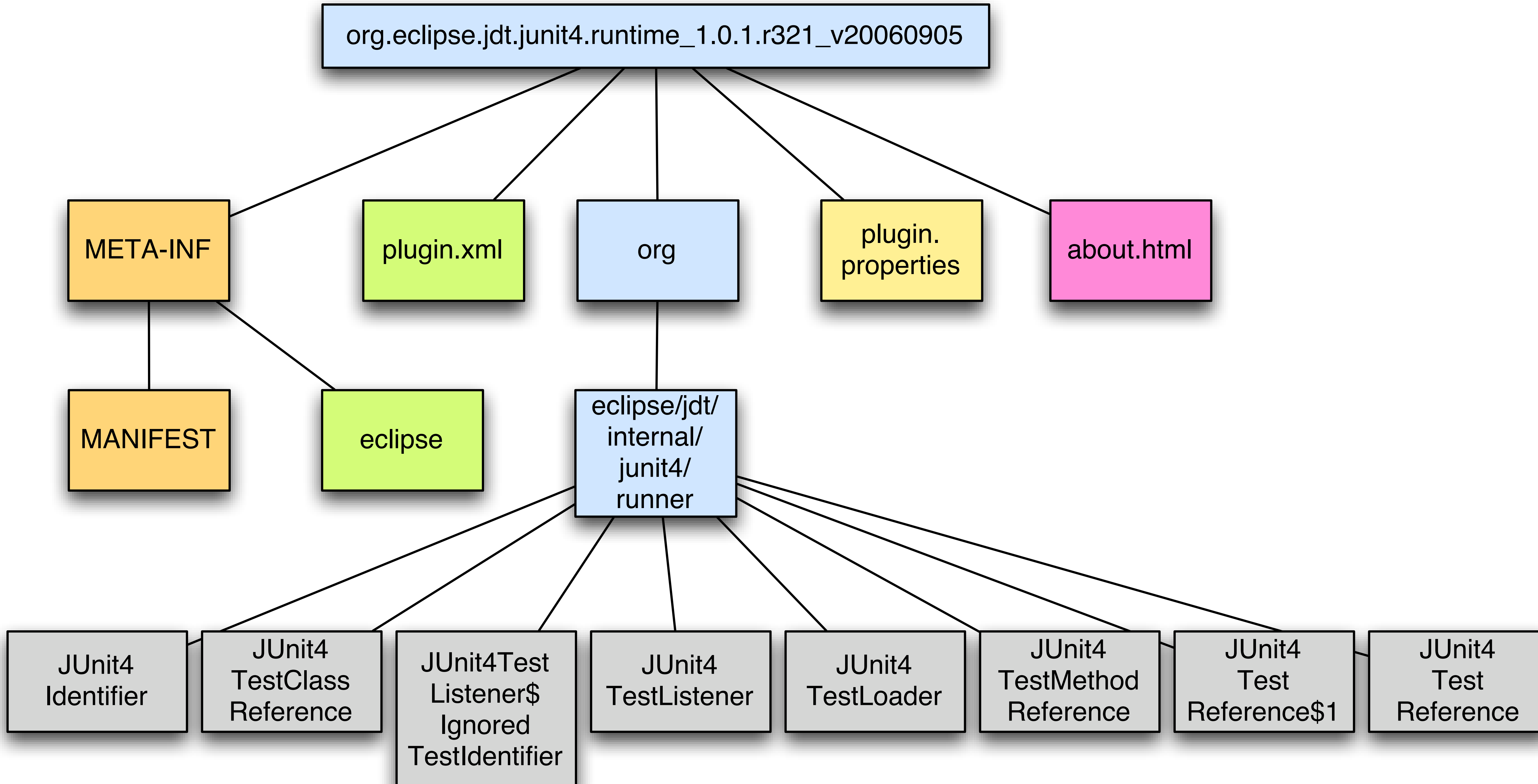
Module Layer: Bundles

(Details regarding native code loading can be found in the OSGi core specification.)

- A bundle is **deployed as a Java ARchive (JAR)** file which contains:
  - resources
    (Including possibly further Jar files; non recursive.)
  - the manifest file
    (META-INF/MANIFEST.MF)
    describing
    - the content
    - how to install and activate the bundle
- After a bundle is started *its services are exposed* to other installed bundles

Module Layer: Bundles

Example of a bundle's content



Life Cycle Layer

API for handling the life-cycle management of applications and components.

An OSGi service platform provides the following functions:

- Install a bundle
- Start / stop a bundle
- Update a bundle
  The OSGi platform stops existing applications, resources are cleaned up, code is unloaded, code is replaced, bundle is restarted.
- Uninstall a bundle
- Monitoring a bundle

Life Cycle Layer

Entities of the OSGi layer.

- **Bundle**
  Represents an installed bundle in the Framework
- **Bundle Context**
  A bundle's execution context within the Framework. The Framework passes this to a Bundle Activator when a bundle is started or stopped. The Bundle Context is used to:
  - access information about the rest of the Framework
  - to install other bundles
  - to access the service registry

Life Cycle Layer

Entities of the OSGi layer.

- **Bundle Activator**
  An **interface implemented by a class** in a bundle that is used to start and stop that bundle.
- **Bundle Event**
  An event that signals a life cycle operation on a bundle. This event is received via a (synchronous) Bundle Listener.
- **Framework Event**
  An event that signals an error or Framework state change. The event is received via a Framework Listener.
- **Bundle Listener**
  A listener to Bundle Events.

Life Cycle Layer

Entities of the OSGi layer.

- **Synchronous Bundle Listener**
  A listener to synchronously delivered Bundle Events.
- **Framework Listener**
  A listener to Framework events.
- **Bundle Exception**
  An Exception thrown when Framework operations fail.
- **System Bundle**
  A bundle that represents the Framework.

Life Cycle Layer

Entities of the OSGi layer.

- Installation of a bundle can only be performed by another bundle (or through implementation specific means.)
- A **Bundle is started through its Bundle Activator**.
- Its Bundle Activator is identified by the Bundle-Activator manifest header. The given class must *implement the* `BundleActivator` *interface* and *provide a default constructor*.

Hello World OSGi Bundle

Implementation of the “main class”.

```
public interface BundleActivator {

    /**
     * Called when this bundle is started so the Framework can perform the
     * bundle-specific activities necessary to start this bundle. This method
     * can be used to register services or to allocate any resources that this
     * bundle needs.
     *
     * This method must complete and return to its caller in a timely manner.
     *
     * @param context The execution context of the bundle being started.
     */
    public void start(BundleContext context) throws Exception;

  ...
```

A not well defined Implementation

Hello World OSGi Bundle

Implementation of the “main class”.

```
public interface BundleActivator {
    ...

    /**
     * Called when this bundle is stopped so the Framework can perform the
     * bundle-specific activities necessary to stop the bundle. In general, this
     * method should undo the work that the <code>BundleActivator.start</code>
     * method started. There should be no active threads that were started by
     * this bundle when this bundle returns. A stopped bundle must not call any
     * Framework objects.
     *
     * <p>
     * This method must complete and return to its caller in a timely manner.
     *
     * @param context The execution context of the bundle being stopped.
     */
    public void stop(BundleContext context) throws Excepti
}
```

The stop method must clean up and stop any running threads

Life Cycle Layer: Bundle Context

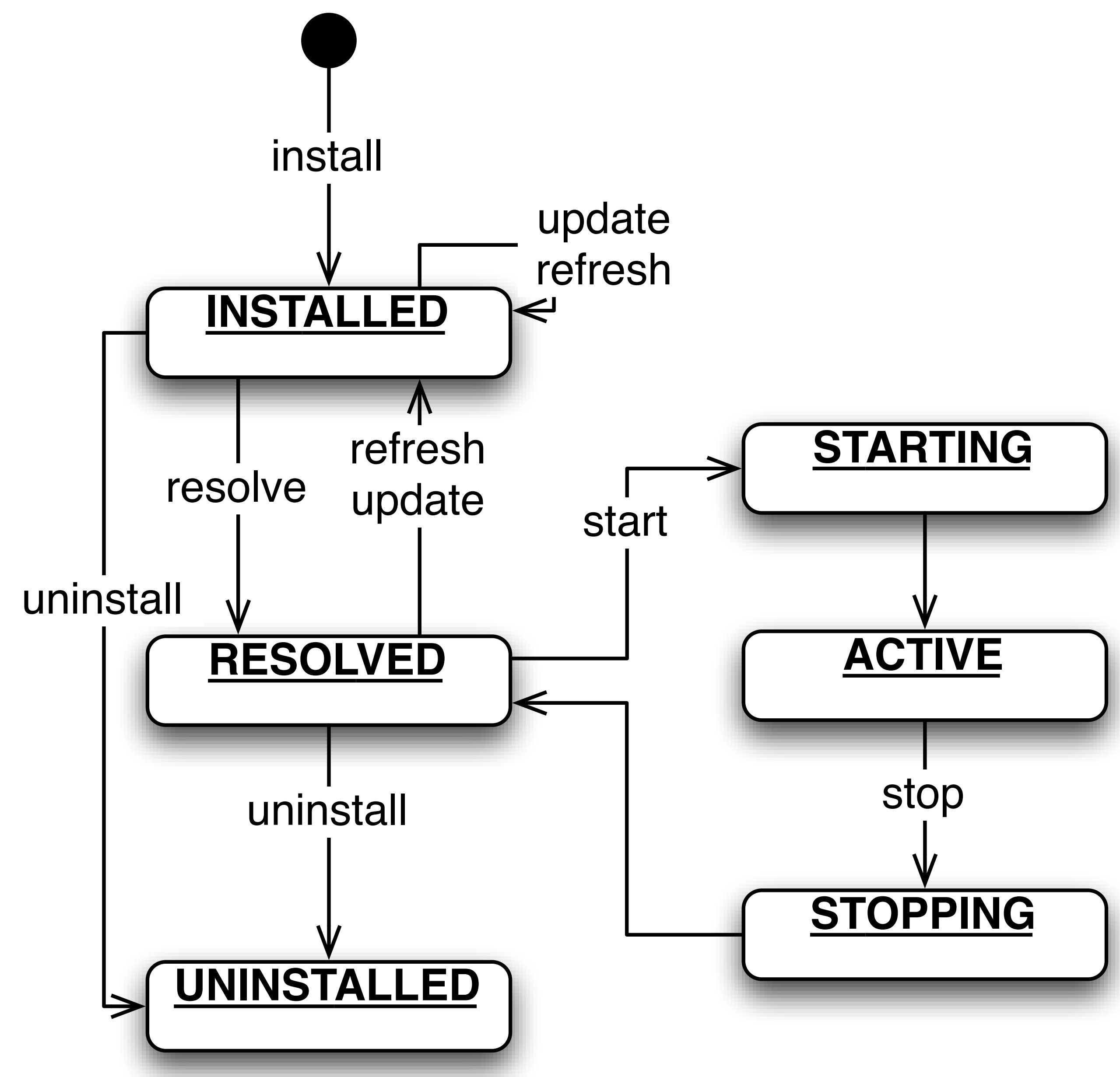
Entities of the OSGi layer.

- **Represents the execution context** of a single bundle; **acts as a proxy** to the underlying framework
- To access a bundle's persistent storage area the BundleContext's `getDataFile(String)` method can be used
  The name is a relative name and translated into an absolute File object, which is then returned.
- The BundleContext interface defines a method for returning information pertaining to framework properties: `getProperty(String)`. E.g. org.osgi.framework.version, org.osgi.framework.vendor, org.osgi.framework.executionenvironment, ...

Life Cycle Layer: Bundle Object

Entities of the OSGi layer.

- For each installed bundle, there is an associated `Bundle` object.
- The **Bundle object** can be **used to manage the bundle's life cycle** and to access reflective information.
  Life cycle methods:
  - `start()`
  - `stop()`
  - `update(...)`
  - `uninstall()`



Life Cycle Layer: System Bundle

Entities of the OSGi layer.

- The **Framework** itself is represented as a bundle
- The bundle representing the **Framework** is referred to as the system bundle
- Through the system bundle, the **Framework** may register services that can be used by other bundles

Life Cycle Layer: Events

Entities of the OSGi layer.

- The `BundleContext`'s methods can be used to add and remove listeners for the following events:
  - `BundleEvent`
    for changes in the life cycle of bundles
  - `FrameworkEvent`
    framework related events, e.g., packages have been refreshed.

Life Cycle Layer: Events

Entities of the OSGi layer

- **Events can be asynchronously delivered**, unless otherwise stated, meaning that they are not necessarily delivered by the same thread that generated the event
- A bundle that **calls a listener should not hold any Java monitors**
Neither the Framework nor the originator of a synchronous event should be in a monitor when a callback is initia

A HelloWorld Bundle

A Very First Example

(Bundle Implementation)

Hello World OSGi Bundle

Implementation of the "main class".

```
package helloworld;

import org.osgi.framework.BundleActivator;
import org.osgi.framework.BundleContext;

public class Activator implements BundleActivator {

    /* ... */

}
```

Here, we have explicit

Hello World OSGi Bundle

Implementation of the “main class”.

```
public class Activator implements BundleActivator {

    public void start(BundleContext context) throws Exception {
        System.out.println("Bundle started: Hello world!");
    }

    public void stop(BundleContext context) throws Exception {
        System.out.println("Bundle stopped: Time to say goodbye.");
    }

}
```

OSGi is based on

Hello World Killer OSGi Bundle

Implementation of the “main class”.

```
public class Activator implements BundleActivator {

   public void start(BundleContext context) {

      System.out.println("HelloWorldKiller searching...");
      Bundle[] bundles = context.getBundles();
      for (int i = 0; i < bundles.length; i++) {
         if ("HelloWorld".equals(bundles[i].getSymbolicName())) {
            try {
               System.out.println("Hello World found, uninstalling!");
               bundles[i].uninstall();
            } catch (BundleException e) {
               System.err.println("Failed: " + e.getMessage());
            } finally { return; }
         }
      }
      System.out.println("Hello World bundle not found");
   }
  ...
}
```

Using the context object to lookup

Module Layer: The Manifest

Selected fields of the manifest used to specify a bundle's properties.

- **Bundle-ManifestVersion**
  for release 4 of the OSGi specification the version is "2"
- **Bundle-Description**
  a short description
- **Bundle-SymbolicName**
  a unique, non-localizable name
- **Bundle-Classpath**
  a comma-separated list of JAR file path names or directories (inside the bundle) containing classes and resources
- **Bundle-Activator**
  specifies the name of the class used to start and stop the bundle

Module Layer: The Manifest

Selected fields of the manifest used to specify a bundle's properties.

- **Bundle-Version**
  the version of this bundle
- **Bundle-RequiredExecutionEnvironment**
  e.g.
  - JRE-1.1,
  - J2SE-1.2, J2SE-1.3, J2SE-1.4, J2SE-1.5,
  - JavaSE-1.6,
  - PersonalJava-1.1, PersonalJava-1.2,
  - CDC-1.0/PersonalBasis-1.0, CDC-1.0/PersonalJava-1.0
- ...(further bundle specific properties)

Module Layer: The Manifest

Fields of the manifest used to specify a bundle's dependencies.

- **Export-Package**
  a declaration of exported packages
- **Import-Package**
  the imported packages for this bundl
- **Require-Bundle**
  specifies the required exports from ar

Enables code protection orthogonal to Java's visibility mechanisms.

OSGi effectively has introduced a new code protection level: if a package in your bundle is not listed on the Export-Package header, then it is only accessible within your module.

Attention: do not import packages that are also defined by your own bundle.

A HelloWorld Bundle

A Very First Example

(The Manifest)

Manifest of a Hello World OSGi Bundle

- The manifest for our HelloWorld Bundle:

Manifest-Version: 1.0
Bundle-Name: HelloWorld
Bundle-Description: A simple hello world bundle.
Bundle-Activator: helloworld.Activator
Import-Package: org.osgi.framework
Bundle-Vendor: Michael Eichberg
Bundle-ManifestVersion: 2
Bundle-SymbolicName: HelloWorld
Bundle-Version: 1.0.0

Should fit in one line!

The class that will be started.

Only required if the bundle interacts with the OSGi runtime.

When specifying the symbolic name it is recommended to follow the guidelines for Java package names.

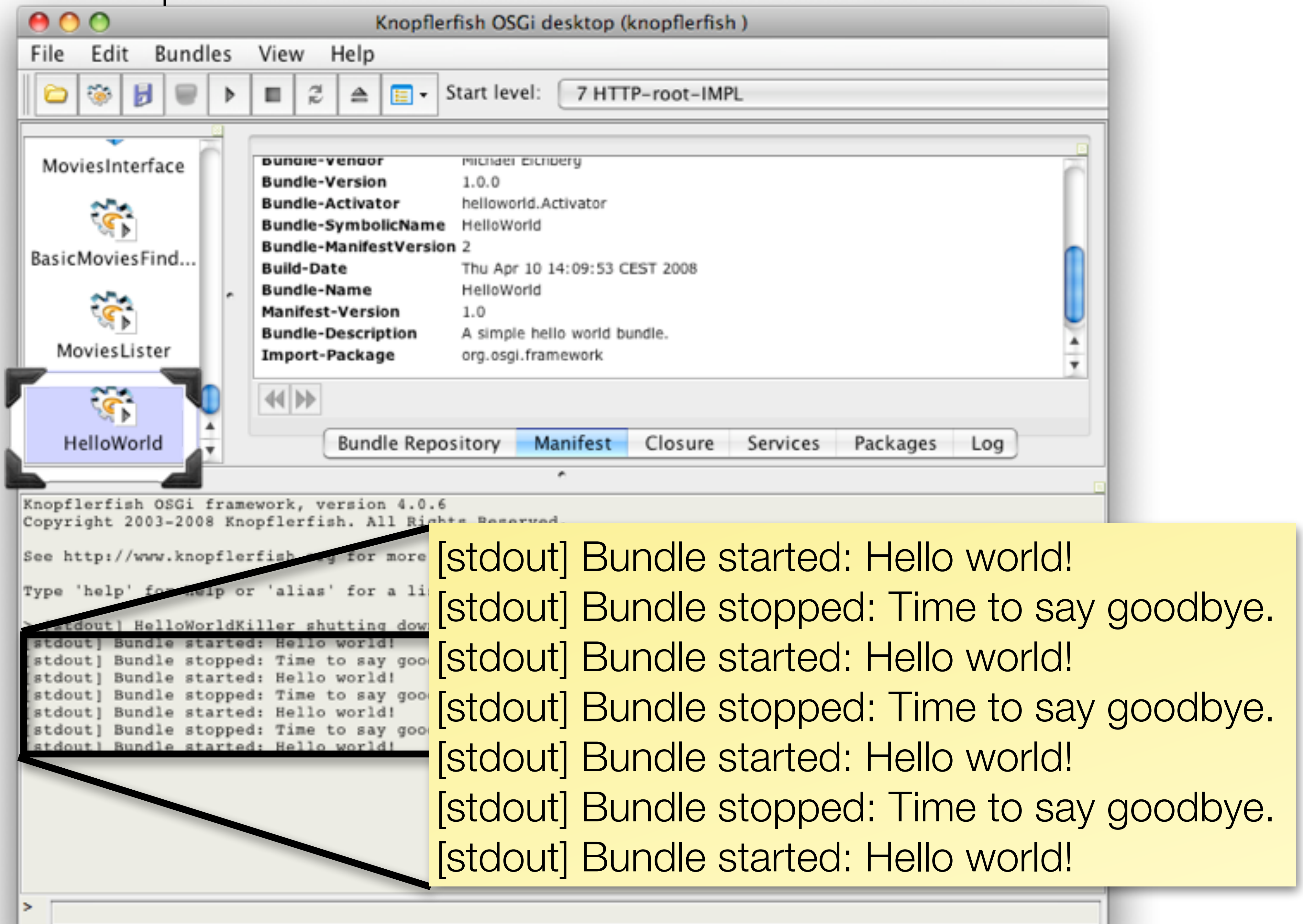
Contents of the Hello World OSGi Bundle

Just the **manifest** and **the class file**.

```
Archive:  OSGi-Rev.1-1.0.0.jar
  Length     Date   Time    Name
 --------    ----   ----    ----
      335  04-10-08 14:09   META-INF/MANIFEST.MF
      841  04-10-08 14:09   helloworld/Activator.class
 --------                   -------
     1176                   2 files
```

Deploying the Hello World OSGi Bundle

Knopflerfish OSGi Desktop



Module Layer: Class Loading

- Bundles can **share a single virtual machine** (VM)
- Within this VM, **bundles can ...**
  - **hide packages and classes** from other bundles
  - **share packages** with other bundles

it is possible that bundle A uses a library L in version X

Module Layer: Class Loading

Bundle Dependencies

- Each bundle is associated with its own class loader that can load classes and resources from:
  - **the boot class path**
  - **framework class path**
  - **bundle space**
    the Jar file that is associated with the bundle and all fragments
  - …

Module Layer: Class Loading

Bundle Dependencies

- Each bundle is associated with its own class loader that can load classes and resources from:
  - ...
  - **class space**
    A class space is all classes reachable from a given bundle's class loader. The space can contain classes from:
    - the parent class loader
    - imported packages
    - required bundles
    - the bundle's class path (private packages)

A standard Singleton.

```java
package demo;
public class MySingleton {

   private static MySingleton instance = null;
   private MySingleton() {}

   public static synchronized MySingleton instance() {
      if (instance == null) instance = new MySingleton();
      return instance;
   }
}
```

```java
package demo;
public class MySingleton {

	private static MySingleton instance = null;
	private MySingleton() {}

	public static synchronized MySingleton instance() {
		if (instance == null) instance = new MySingleton();
		return instance;
	}
}
```

Code

Output

```java
Object a = MySingleton.instance();
Object b = MySingleton.instance();
System.out.println(a == b);
```

```
true
```

```java
package demo;
public class MySingleton {

	private static MySingleton instance = null;
	private MySingleton() {}

	public static synchronized MySingleton instance() {
		if (instance == null) instance = new MySingleton();
		return instance;
	}
}
```

Code

```java
ClassLoader cl1 = ClassLoader.getSystemClassLoader();
Class<?> clazz1 = cl1.loadClass("demo.MySingleton");
Object a = clazz1
   .getDeclaredMethod("instance", new Class<?>[] {})
      .invoke(null);
ClassLoader cl2 = ClassLoader.getSystemClassLoader();
Class<?> clazz2 = cl2.loadClass("demo.MySingleton");
Object b = clazz2
   .getDeclaredMethod("instance", new Class<?>[] {})
      .invoke(null);

System.out.println(a == b);
```

Output

true

```java
package demo;
public class MySingleton {

	private static MySingleton instance = null;
	private MySingleton() {}

	public static synchronized MySingleton instance() {
		if (instance == null) instance = new MySingleton();
		return instance;
	}
}
```

Code

Output

```java
ClassLoader cl1 = new MyClassLoader();
Class<?> clazz1 = cl1.loadClass("demo.MySingleton");
Object a = clazz1
   .getDeclaredMethod("instance", new Class<?>[] {})
     .invoke(null);
ClassLoader cl2 = new MyClassLoader();
Class<?> clazz2 = cl2.loadClass("demo.MySingleton");
Object b = clazz2
   .getDeclaredMethod("instance", new Class<?>[] {})
     .invoke(null);

System.out.println(a == b);
```

???

Module Layer: Class Loading

Resolving Bundle Dependencies

- Resolving is the process where importers are wired to exporters
- Resolving is a process of satisfying constraints
- Resolving must take place before any code from a bundle can be loaded or executed

Module Layer: Class Loading

Resolving Bundle Dependencies

Resolving is the process where importers are wired to exporters.

- Constraints on the wires are statically defined by:
  - Import and export packages
  - Required bundles, which import all exported packages from a bundle
  - Fragments, which provide their contents and definitions to the host
- A bundle can be resolved if the following conditions are met:
  - All its mandatory imports are wired
  - All its mandatory required bundles are available and their exports wired

Module Layer: Class Loading

Resolving Bundle Dependencies; Mechanisms to Match Imports to Exports

Version Matching

- Bundle A:
  `Import-Package: p; version="[1,2)"`
- Bundle B:
  `Export-Package: p; version=1.5.1`

resolves correctly

Module Layer: Class Loading

Resolving Bundle Dependencies; Mechanisms to Match Imports to Exports

Optional Packages

A bundle can indicate that it does not require a package to resolve correctly, but it may use the package if it is available.
For example, logging is important, but the absence of a log service should not prevent a bundle from running.

- Bundle `A`:
  `Import-Package: p; resolution:=optional; version=1.6`
- Bundle `B`:
  `Export-Package: p; q; version=1.5.0`

If you specify an optional dependency your code must be

resolves correctly, but the package p is not available to `A` due to version conflicts.

Module Layer: Class Loading

Resolving Bundle Dependencies; Mechanisms to Match Imports to Exports

Package Constraints

Classes can depend on classes in other packages. These inter-package dependencies are modeled with the uses directive on the Export-Package header.
Example:

- Bundle A:
  ```
  Import-Package: q; version="[1.0,1.0]"
  Export-Package: p; uses:="q"
  ```
- Bundle B:
  ```
  Export-Package: q; version=1.0
  ```

can be resolved.

Module Layer: Class Loading

Resolving Bundle Dependencies; Mechanisms to Match Imports to Exports

Package Constraints

- Bundle A

```
package org.bar.q;
...
   public org.foo.common.p.PType someMethod() {...}
...

import-package: org.foo.common.p
export-package: org.bar.q,uses:="org.foo.common.p"
```

Record the "leakage" - this information is required by the resolver to make sure that the packages are correctly wired.

Module Layer: Class Loading

Resolving Bundle Dependencies; Mechanisms to Match Imports to Exports

- **Attribute Matching**
  Allows the importer and exporter to influence the matching process in a declarative way.
- **Class Filtering**
  limits the visibility of the classes in a package with the include and exclude directives on the export def
- **Provider selection**
  allows the importer to select which bundles can be considered as exporters.

Module Layer: Class Loading

Resolving Bundle Dependencies

Runtime Class Loading

- **After a bundle is resolved**, the Framework creates **one class loader for each bundle** that is not a fragment
- This class loader provides **each bundle with its own name space**, to avoid name conflicts, and allows resource sharing with other bundles

Module Layer: Fragments

Resolving Bundle Dependencies

A fragment allows to supply entries that are **inserted into the host's Bundle-Classpath**. The following example illustrates this:

- Bundle A:

```
Bundle-SymbolicName: A
Bundle-Classpath:
required.jar,optional.jar,default.jar,.
```

- Bundle B:

```
Bundle-SymbolicName: B
Bundle-Classpath: fragment.jar
Fragment-Host: A
```

The bundle-classpath must include the directory (within the bundle) that contains the bundles class files.

In these examples, the bundle itself is not in the classpath. In general "." is also part of the Bundle-Classpath (see http://www.aqute.biz/Blog/2007-02-19)

Module Layer: Bundle Class Path

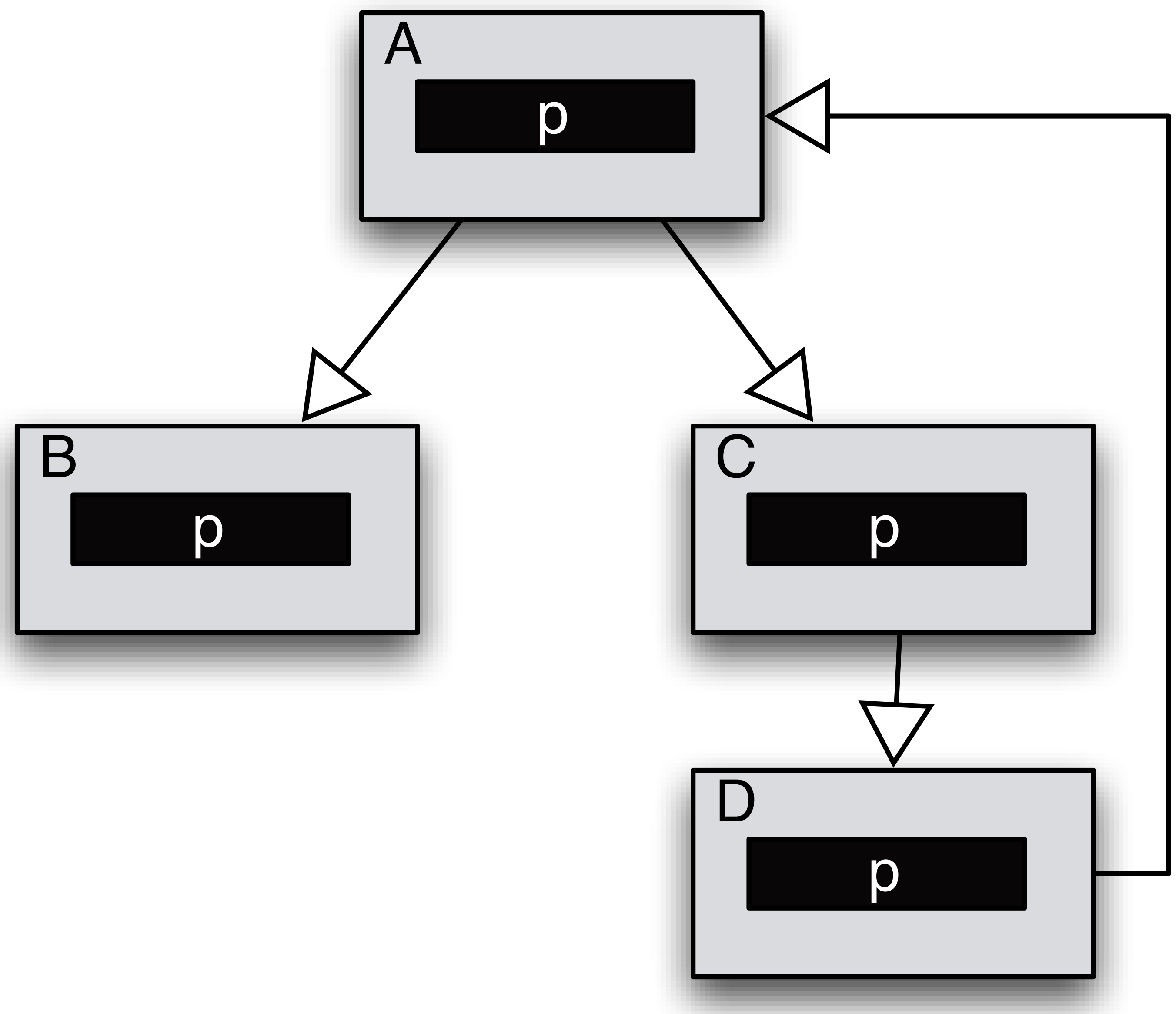
Resolving Bundle Dependencies

- Intra bundle class path dependencies are declared in the **Bundle-Classpath** manifest header
- It declares the bundle's embedded class path using one or more JAR files or directories that are contained in the bundle's JAR file
- When locating a class path entry in a bundle, the Framework must attempt to locate the class path entry relative to the root of the bundle's JAR.
  If a class path entry cannot be located in the bundle, then the Framework must attempt to locate the class path entry in each of the attached fragment bundles.

Module Layer: Cyclic Bundle Dependencies

Locating Resources (Classes)

- OSGi uses a depth first search order in case of cyclic dependencies.
  - Bundle A: `Require-Bundle: B, C`
  - Bundle B: `“No Requirements”`
  - Bundle C: `Require-Bundle: D`
  - Bundle D: `Require-Bundle: A`
- Resulting bundle search order: B, D, C, A.



Module Layer: Bundle Dependencies

Design Guideline:

The preferred way of **wiring bundles is to use the Import-Package and Export-Package** headers because they couple the importer and exporter to a much lesser extent than using require bundle.

Example Bundles and Fragments

org.eclipse.jdt.junit4.runtime

```
Manifest-Version: 1.0

Bundle-RequiredExecutionEnvironment: J2SE-1.5
Bundle-ManifestVersion: 2
Bundle-Localization: plugin
Bundle-SymbolicName: org.eclipse.jdt.junit4.runtime
Require-Bundle: org.junit4;bundle-version="[4.1.0,4.2.0)",
org.eclipse.jdt.junit.runtime;bundle-version="[3.2.0,4.0.0)"
Export-Package:  org.eclipse.jdt.internal.junit4.runner; x-internal:=true
Bundle-Version: 1.0.1.r321_v20060905

Eclipse-LazyStart: true
```

The Bundle-Localization header contains the location in the bundle where

Example Bundles and Fragments

org.eclipse.core.filesystem.macosx

```
Manifest-Version: 1.0

Bundle-ManifestVersion: 2
Fragment-Host: org.eclipse.core.filesystem;bundle-version="[1.0.0,2.0.0)"
Bundle-Localization: fragment
Bundle-SymbolicName: org.eclipse.core.filesystem.macosx; singleton:=true
Bundle-Version: 1.0.0.v20060603

Eclipse-PlatformFilter: (& (osgi.os=macosx) (|(osgi.arch=x86) (osgi.arch=ppc)))
```

Service Layer

Supporting Loosely Coupled Application Designs

- The OSGi Service Platform provides a lightweight **publish, find and bind service model** for services inside the JVM with the OSGi Framework service registry
- A service allows one bundle to provide functionality to other bundles
- A service is a **normal Java object (the service object) that is registered under one or more Java interfaces (the service interfaces)** with the service registry
- Bundles can register services, search for them, or receive notifications when their registration state changes
- When a bundle is stopped, all the services registered with the Framework by a bundle must be automatically unregistered

Service Layer: Functionality

Supporting Loosely Coupled Application Designs

- Full access to the Service Layer’s internal state is provided (Reflective)
- Access to services can be restricted (Secure)

Service Layer: Entities

Supporting Loosely Coupled Application Designs

- **Service**
  An object registered with the service registry under one or more interfaces together with properties. This object can be discovered and used by bundles.
  The service object is owned by, and runs within, a bundle.
- **Service Registry**
  Holds the service registrations.
- **Service Reference**
  A reference to a service. Provides access to the service's properties but not the actual service object. The service object must be acquired through a bundle's Bundle Context.

Service Layer: Entities

Supporting Loosely Coupled Application Designs

- **Service Registration**
  The receipt provided when a service is registered. The service registration allows the update of the service properties and the unregistration of the service.
- **Service Permission**
  The permission to use an interface name when registering or using a service.
- **Service Factory**
  A facility to let the registering bundle customize the service object for each using bundle.
- **Service Listener**
  A listener to Service Events.

Service Layer: Entities

Supporting Loosely Coupled Application Designs

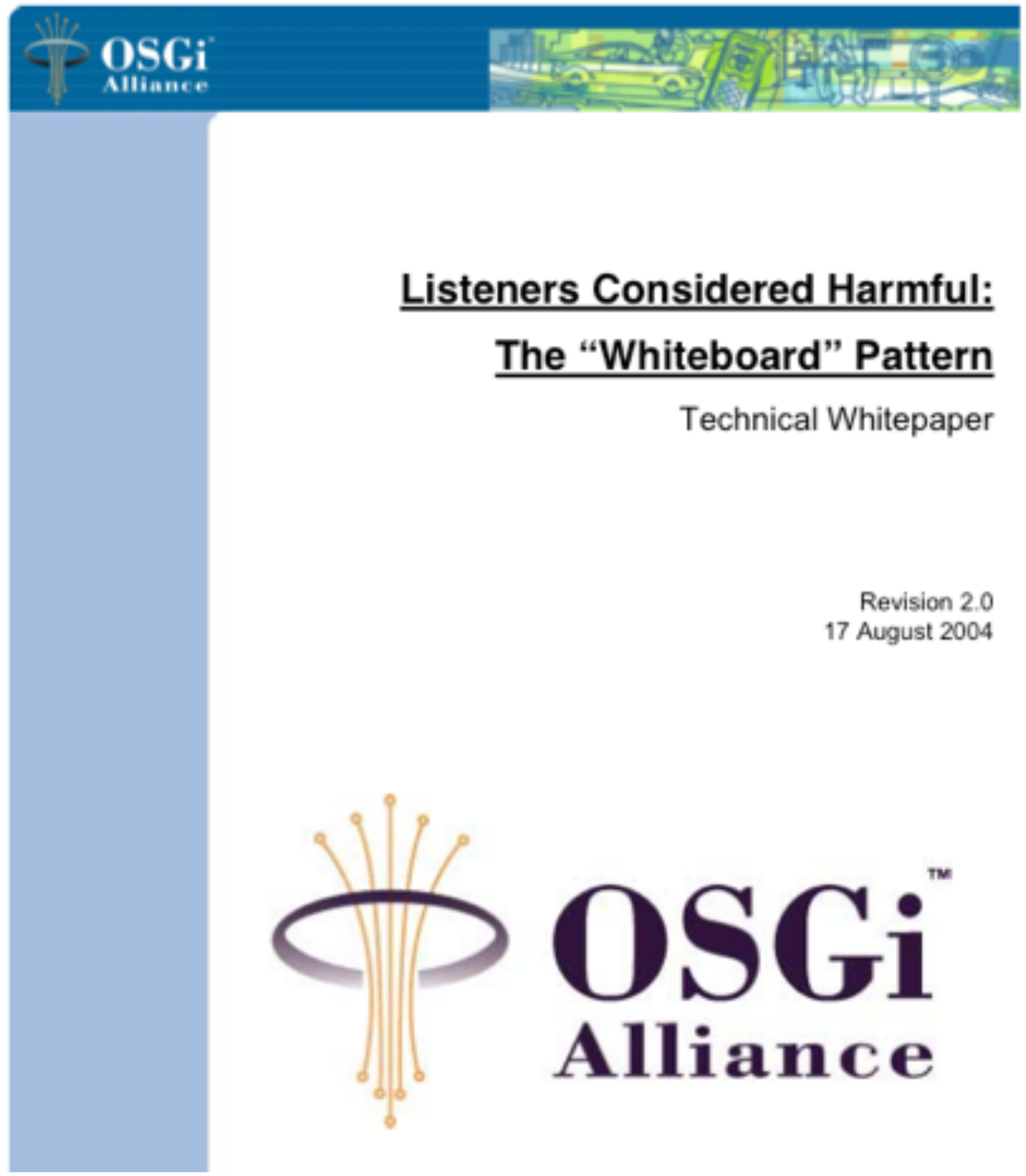
- **Service Event**
  An event holding information about the registration, modification, or unregistration of a service object.
- **Filter**
  An object that implements a simple but powerful filter language. It can select on properties.

The Whiteboard Pattern

Excursion

OSGi

background information



Listeners Considered Harmful:

The "Whiteboard" Pattern

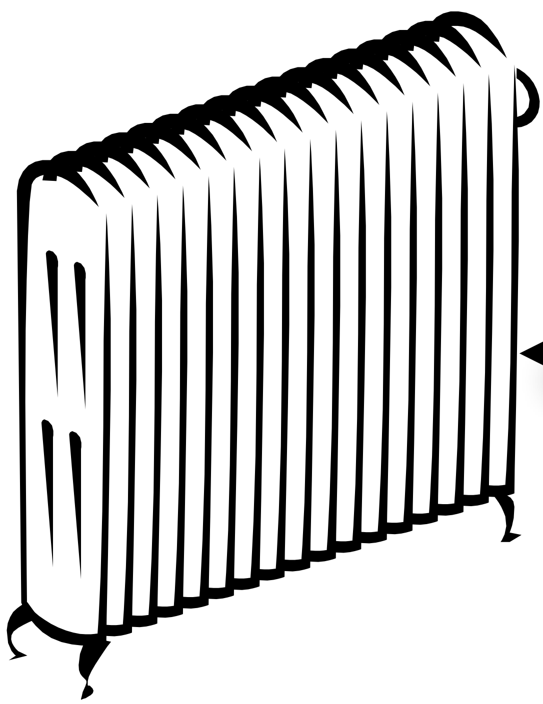
Technical Whitepaper

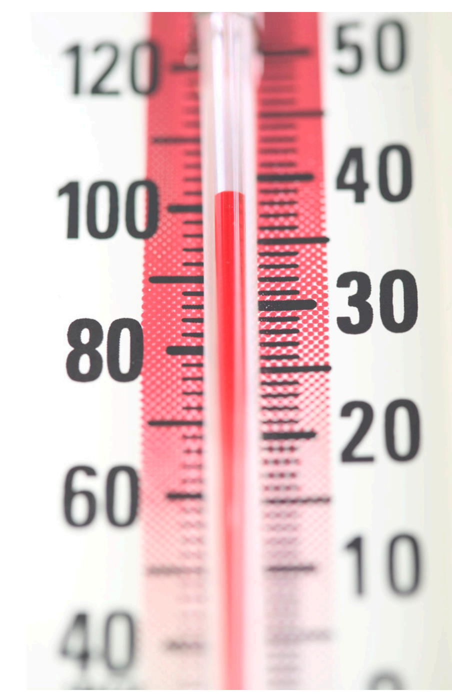
Revision 2.0
17 August 2004

www.osgi.org/wiki/uploads/Links/whiteboard.pdf

Example Smart Home Scenario

Temperature
Sensor





Radiator

How could an implementation look like?

The Observer Pattern

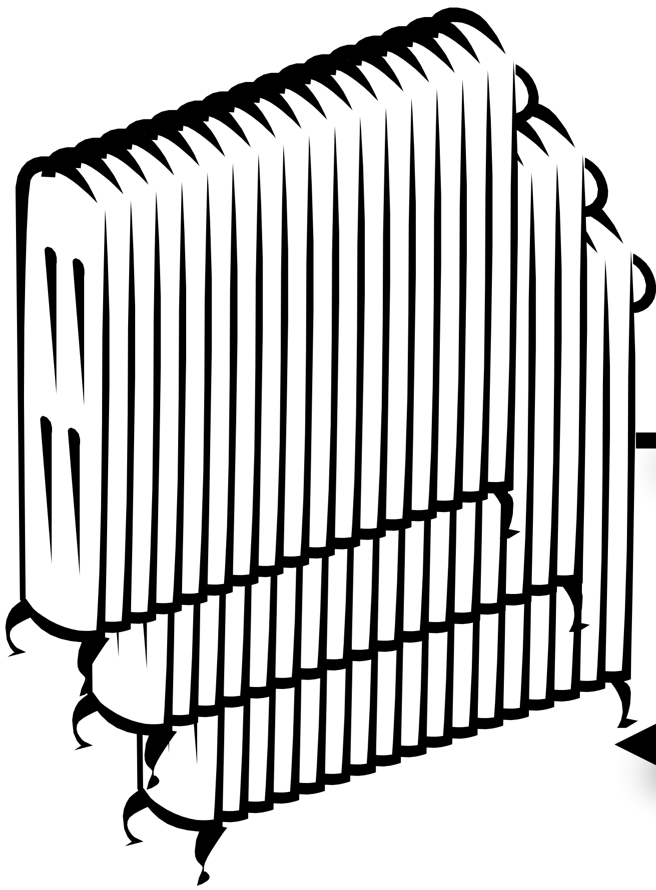
Class Diagram

- Intent:
  Define a **one-to-many dependency between objects** so that when object changes state, all its dependents are notified and updated automatically.
- ...

Example Smart Home Scenario

Temperature Sensor

1: register as listener



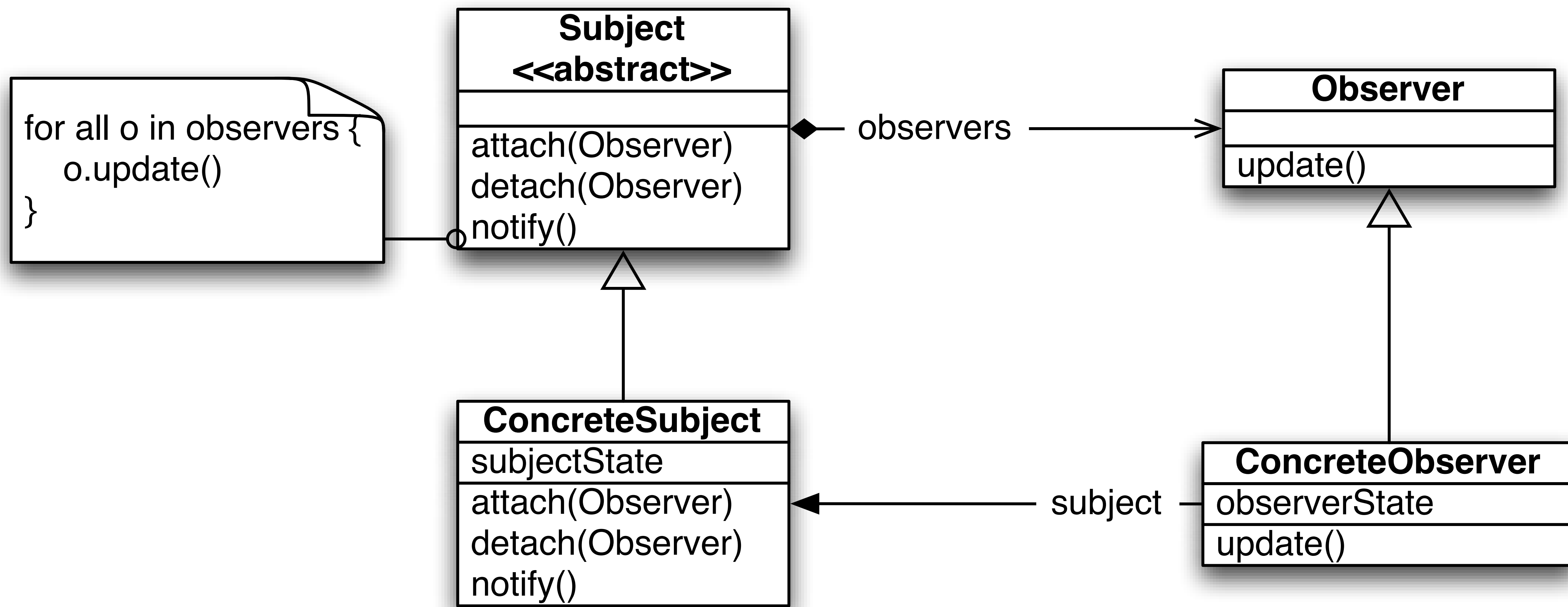
2: signal temperature changes

Radiator

How could an implementation look like?

The Observer Pattern

Class Diagram



[Design Patterns; Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides, Addison Wesley 1995]

Sensor Enables the Registration of “Listeners”

First Idea

```
public interface TempListener {

    public void tempChanged(TempChan
}


public class TemperatureSensor {

    private final Set<TempListener> tempListeners = new HashSet<TempListener>();

    public void registerTempListener(TempListener tl) {
        tempListeners.add(tl);
    }

    public void unregisterTempListener(TempListener tl) {
        tempListeners.remove(tl);
    }
    ...
}
```

Temp = Temperature

Sensor Enables the Registration of “Listeners”

First Idea

```
public class Radiator implements TempListener {

    private final TemperatureSensor sensor;

    public Radiator(TemperatureSensor sensor) {
        sensor.registerTempListener(this);
        this.sensor = sensor;
    }

    public void dispose() {
        sensor.unregisterTempListener(this);
    }

    public void tempChanged(TempChangedEvent event) {
        ...
    }
}
```

IoC, Dependency Injection and OSGi...:
Many containers have been developed, for example PicoContainer, HiveMind, Spring, and even EJB 3.0.

However there is one limiting factor of all these containers to date: they are mostly static. Once a TemperatureSensor is given to a Radiator , it tends to be associated for the lifetime of the JVM.

The Observer Pattern

Problems in the Context of OSGi

- Problems with the Observer Pattern in *continuously running and dynamic applications* (e.g. SmartHome scenario):
  - **When the event source goes away the observer must clean up any references it holds.**
  - **When the observer goes away, the event source (subject) should remove it from the list of observers.**

- In an OSGi environment, the owner of an object can and will go away.

Dependencies and Stale References

Implementation Restriction

Bundles must listen to events generated by the Framework to clean up and remove *stale references*.

- A stale reference is a reference to a Java object that belongs to the class loader of a bundle that is stopped or is associated with a service object that is unregistered.
- It has to be ensured that stale references are deleted.

The Whiteboard

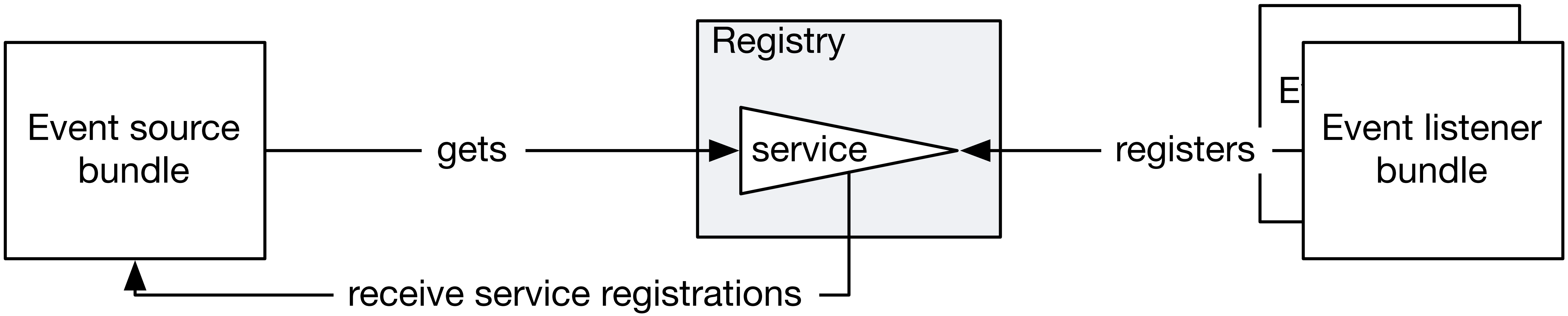
Outline

- Goal:
  No private registries as required by the observer pattern.

- Description:
  - Each **event listener registers itself** as a service (e.g. `HeatingSystem`) with the OSGi service registry.
  - When the event source (e.g. `TemperatureSensor`) has an event object to deliver, the event source calls all event listeners (e.g. the `HeatingSystem` service) in the service registry. Hence, the inter-bundle dependencies between the event source and the event listener is handled by the framework.

The Whiteboard

Structure



[Listeners Considered Harmful: The “Whiteboard” Pattern; Revision 2.0; OSGi Alliance , 17. August 2004]

A Movie Finder Service

A Movie Finder Service

1. Bundle
MovieFinder Service - **Interface**

MovieFinder Service - Interface

API

```java
package movies;

public interface MovieFinder {

    Movie[] findAll();
}
```

MovieFinder Service - Interface

API

```
package movies;

public class Movie {

    private final String title;
    private final String director;

    public Movie(String title, String director) {
        this.title = title;
        this.director = director;
    }

    public String getTitle() {
        return title;
    }

    public String getDirector() {
        return director;
    }
}
```

MovieFinder Service - Interface

Metadata

```
Manifest-Version: 1.0
Bundle-Name: MoviesInterface
Bundle-Description: Declaration of an interface to find movies.
Bundle-Vendor: Michael Eichberg
Bundle-ManifestVersion: 2
Bundle-SymbolicName: MoviesInterface
Bundle-Version: 1.0.0
Export-Package: movies;specification-version=1.0.0
```

A Movie Finder Service

2. Bundle
MovieFinder Service - **Implementation**

MovieFinder Service - Implementation

Sourcecode

```
package movies.spi;

import movies.Movie;
import movies.MovieFinder;

public class BasicMovieFinder implements MovieFinder {

    private static final Movie[] MOVIES = new Movie[] {
            new Movie("The Godfather", "Francis Ford Coppola"),
            new Movie("Spirited Away", "Hayao Miyazaki")
    };

    public Movie[] findAll() {

        return MOVIES;
    }
}
```

MovieFinder Service - Implementation

Sourcecode

```java
package movies.spi;
import ...;

public class BasicMoviesFinderActivator implements BundleActivator {

    private ServiceRegistration registration;

    public void start(BundleContext context) {
        MovieFinder finder = new BasicMovieFinder();
        registration = context.registerService(
                MovieFinder.class.getName(),
                finder,
                new Properties());
    }

    public void stop(BundleContext context) {
        registration.unregister();
    }
}
```

Registration of the service

Unregistration of the service (avoid stale references)

MovieFinder Service - Implementation

Metadata

```
Manifest-Version: 1.0
Bundle-Name: BasicMoviesFinderService
Bundle-Description: Implementation of a movie finder service.
Import-Package: movies;version="[1.0.0,2.0.0)",org.osgi.framework
Bundle-Vendor: Michael Eichberg
Bundle-ManifestVersion: 2
Bundle-SymbolicName: MoviesFinderServiceSPI
Bundle-Version: 1.0.0
Bundle-Activator: movies.spi.BasicMoviesFinderActivator
```

Using the Movie Finder Service

3. Bundle

Implementation of a *MovieLister* Service that uses the *MovieFinder* Service

MovieLister Service - Interface

API

```
package movies.lister;

import java.util.List;
import movies.Movie;


public interface MovieLister {

    List<Movie> listByDirector(String name);
}
```

MovieLister Service - Implementation

Sourcecode

```
package movies.lister.spi;

public class MovieLister implements movies.lister.MovieLister {

    private final Collection<MovieFinder> finders = Collections
            .synchronizedCollection(new ArrayList<MovieFinder>());

    protected void bindFinder(MovieFinder finder) {
        finders.add(finder);
        System.out.println("MovieLister: added a finder");
    }

    protected void unbindFinder(MovieFinder finder) {
        finders.remove(finder);
        System.out.println("MovieLister: removed a finder");
    }

    ...
```

Handle dynamic service (un)registration.

MovieLister Service - Implementation

Sourcecode

```
package movies.lister.spi;

public class MovieLister implements movies.lister.MovieLister {

    ...

    public List<Movie> listByDirector(String director) {
        MovieFinder[] finderArray = finders.toArray(new MovieFinder[finders.size()]);
        List<Movie> result = new LinkedList<Movie>();
        for (int j = 0; j < finderArray.length; j++) {
            Movie[] all = finderArray[j].findAll();
            for (int i = 0; i < all.length; i++) {
                if (director.equals(all[i].getDirector())) {
                    result.add(all[i]);
                }
            }
        }
        return result;
    }
}
```

Use all MovieFinder services.

MovieLister Service - Implementation

Sourcecode

```
public class MovieFinderTracker extends ServiceTracker {

    private final MovieLister lister = new MovieLister();

    private int finderCount = 0;

    private ServiceRegistration registration = null;

    public MovieFinderTracker(BundleContext context) {

        super(context, MovieFinder.class.getName(), null);
    }

    private boolean registering = false;
    ...
```

MovieLister Service - Implementation

Sourcecode

```
public class MovieFinderTracker extends ServiceTracker {
    ...

    @Override public Object addingService(ServiceReference reference) {
        MovieFinder finder = (MovieFinder) context.getService(reference);
        lister.bindFinder(finder);
        synchronized (this) {
            finderCount++;
            if (registering) return finder;
            registering = (finderCount == 1);
            if (!registering) return finder;
        }
        ServiceRegistration reg = context.registerService(MovieLister.class
                .getName(), lister, null);

        synchronized (this) { registering = false; registration = reg; }
        return finder;
    }
    ...
```

registering = (finderCount ==1)
⇢ make the MovieLister service available, if the service is not yet available

MovieLister Service - Implementation

Sourcecode

```java
public class MovieFinderTracker extends ServiceTracker {
    ...
    @Override public void removedService(
            ServiceReference reference,Object service) {
        MovieFinder finder = (MovieFinder) service;
        lister.unbindFinder(finder);
        context.ungetService(reference);
        ServiceRegistration needsUnregistration = null;
        synchronized (this) {
            finderCount--;
            if (finderCount == 0) {
                needsUnregistration = registration;
                registration = null;
            }
        }
        if (needsUnregistration != null) {
            needsUnregistration.unregister();
        }
    }
}
```

MovieLister Service - Implementation

Sourcecode

```java
package movies.lister.spi;

import org.osgi.framework.BundleActivator;
import org.osgi.framework.BundleContext;

public class MovieListerActivator implements BundleActivator {

    private MovieFinderTracker tracker;

    public void start(BundleContext context) {
        tracker = new MovieFinderTracker(context);
        tracker.open();
    }

    public void stop(BundleContext context) {
        tracker.close();
    }
}
```

MovieFinder Service - Implementation

Metadata

```
Manifest-Version: 1.0
Bundle-Name: MoviesListerDynamicService
Bundle-Description: Implementation of a movie lister service.
Import-Package: org.osgi.framework, org.osgi.util.tracker, movies
Bundle-Vendor: Michael Eichberg
Bundle-ManifestVersion: 2
Bundle-SymbolicName: MoviesListerDynamicServiceSPI
Bundle-Version: 1.0.0
Bundle-Activator: movies.lister.spi.MovieListerActivator
Export-Package: movies.lister
```