Architektur iterativ auf Basis von OSGi entwickeln

Ein Vortrag von

Sven Jeppsson (syngenio AG)

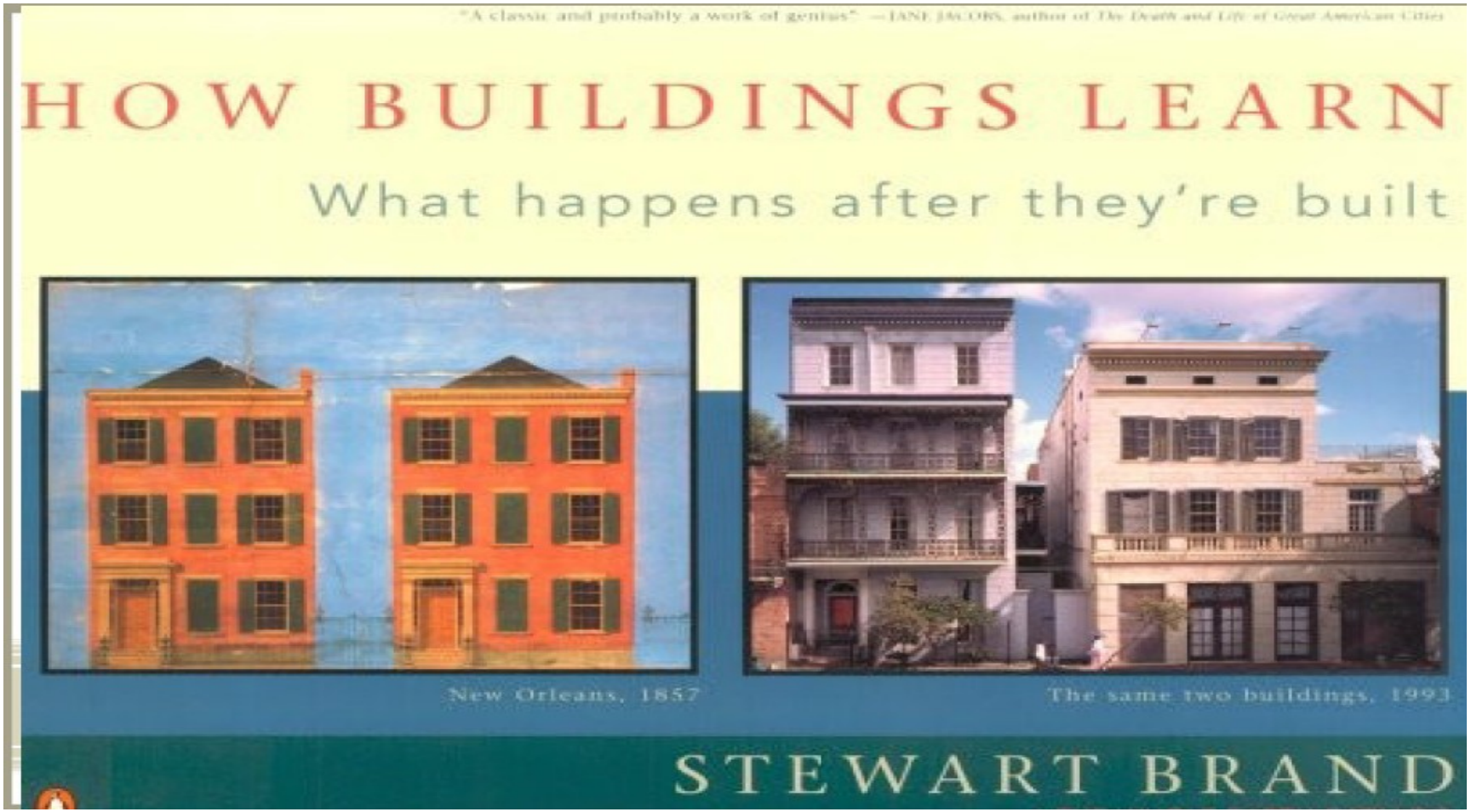
und

Karsten Panier (Signal Iduna Gruppe)

Inhalt

- Motivation
- Architektur
- Architektur Evolution
- OSGi
- Refactoring Beispiel
- Migration bestehender Systeme
- Probleme mit OSGi
- Fazit

Architektur ist im Wandel



http://javamagazin.de/mediapool/podcasts/java/jax_link/jax_link_PAF4.pdf

Motivation

- Umfeld und Anforderungen laufend im Wandel
- ➔ ***Also Software im Wandel***

- Agile Methoden tragen dem Rechnung
- Widerspricht einer Front-Up Architektur
- ➔ ***Architektur ist evolutionär!***

Architektur Evolution

Was ist Architektur Evolution?

Was ist Architektur?

Architektur Definition

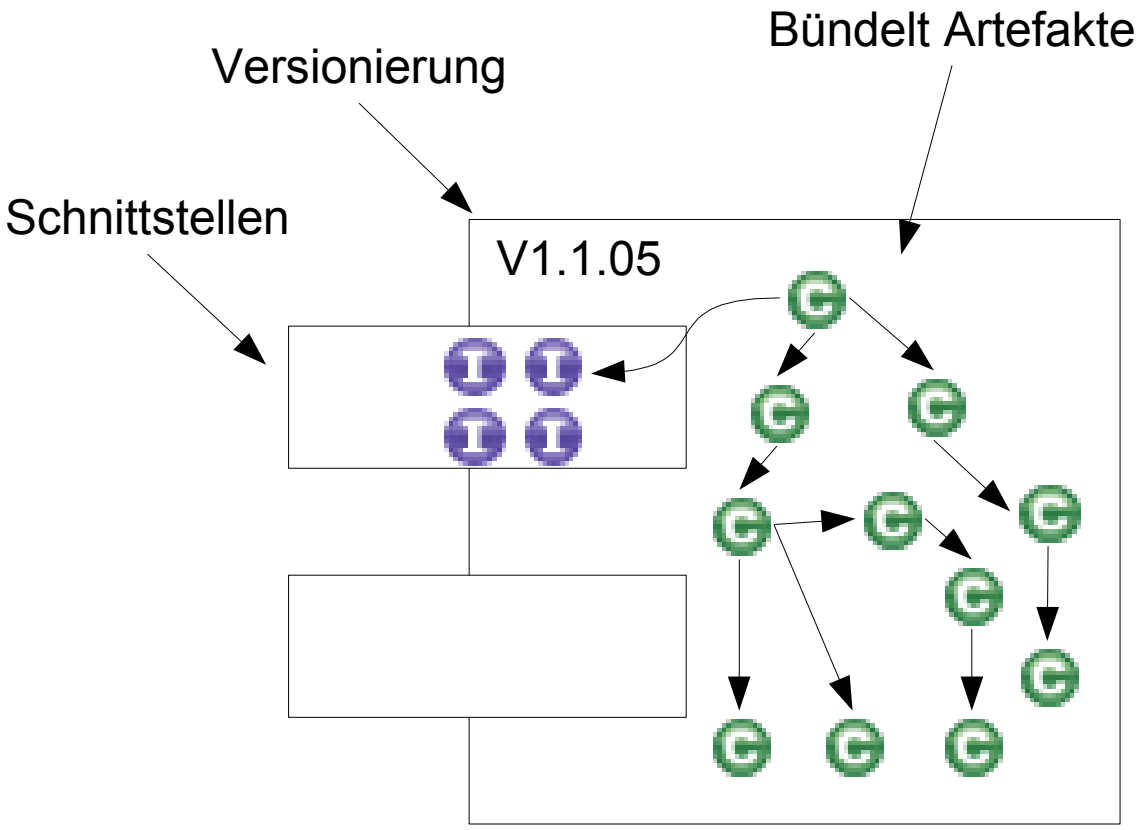
„eine strukturierte oder hierarchische Anordnung der Systemkomponenten, sowie Beschreibung ihrer Beziehungen“

(Lit.: Balzert, S. 716).

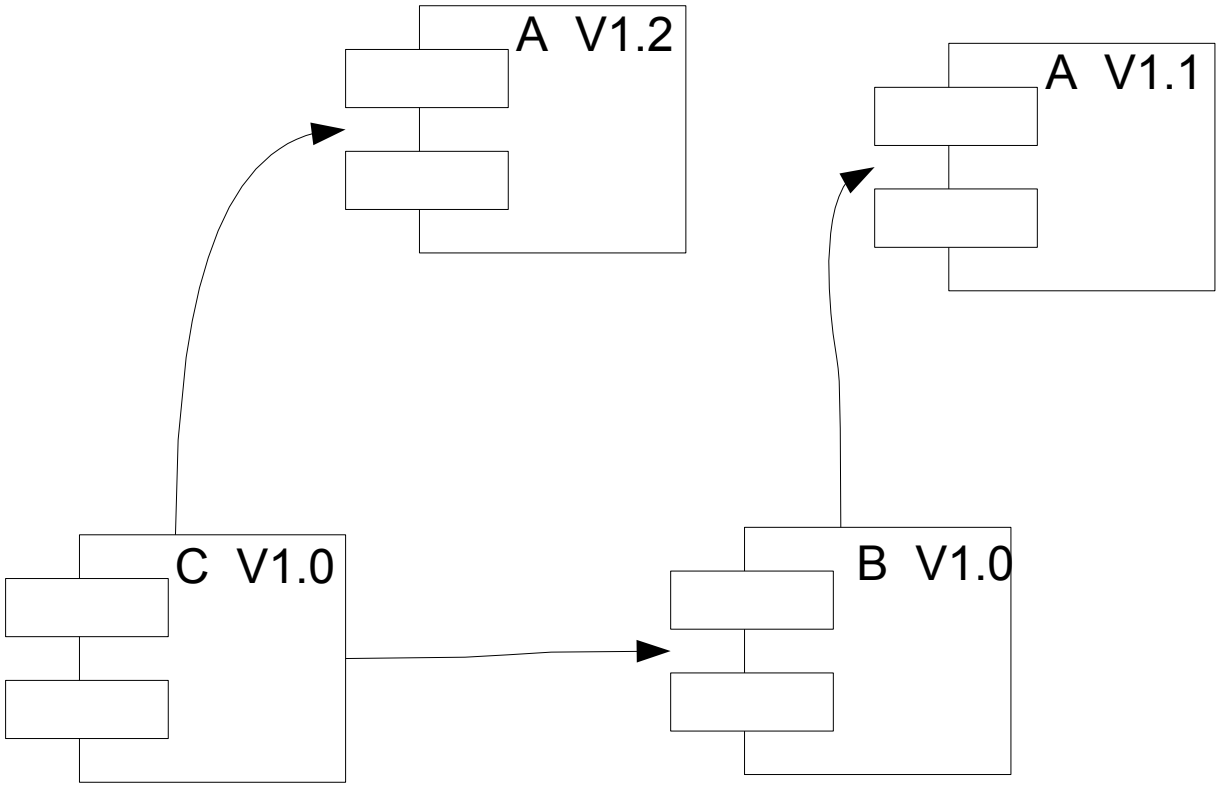
Architektur Definition II

- Was ist eine Komponente?
- Was sind Beziehungen zwischen Komponenten?

Komponente



Beziehungen zwischen Komponenten



Lebenszyklen einer Komponente

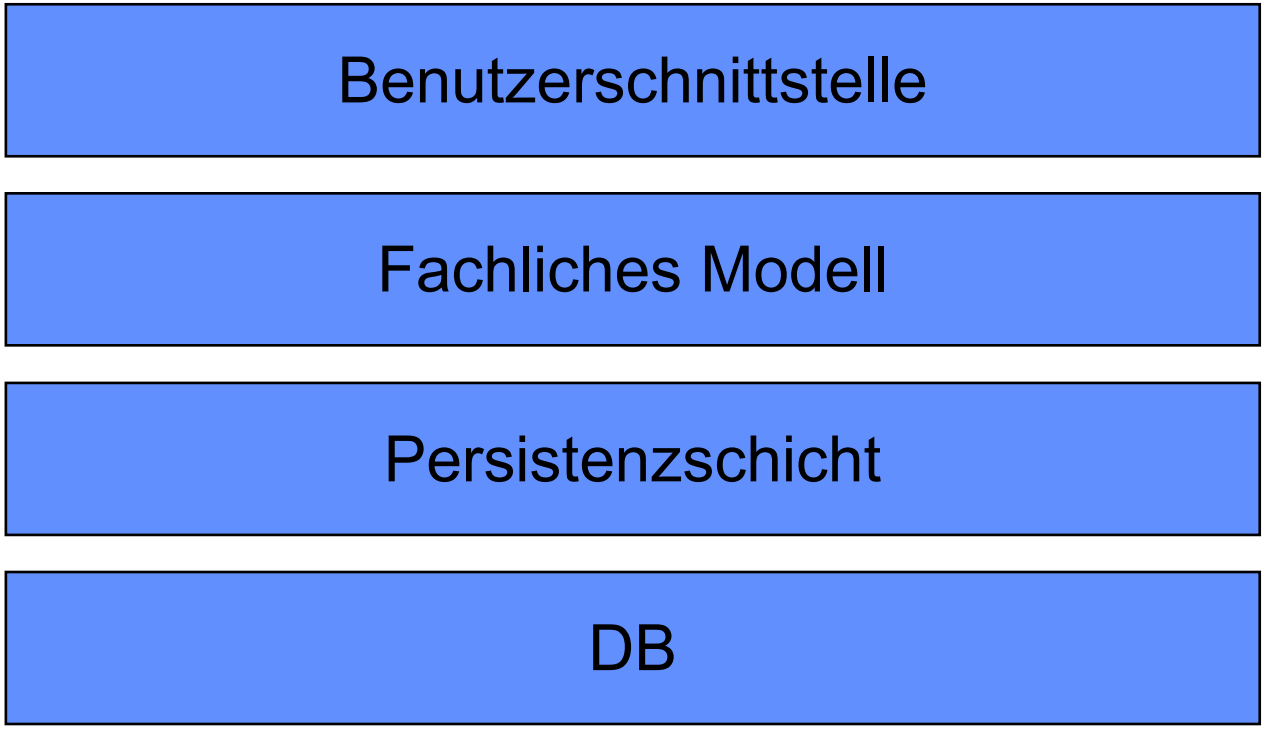
- Statischer Lebenszyklus
  - Versionen der Komponente
    - freigegebene Versionen
  - Wiederverwendbarkeit
    - Einsatz in verschiedenen Systemen
- Dynamischer Lebenszyklus
  - Zustände im Ablauf des Systems
    - installiert, gestartet
  - Auflösen von Abhängigkeiten zu anderen Komponenten (Version)

Schneiden von Komponenten

Was gehört in eine Komponente?

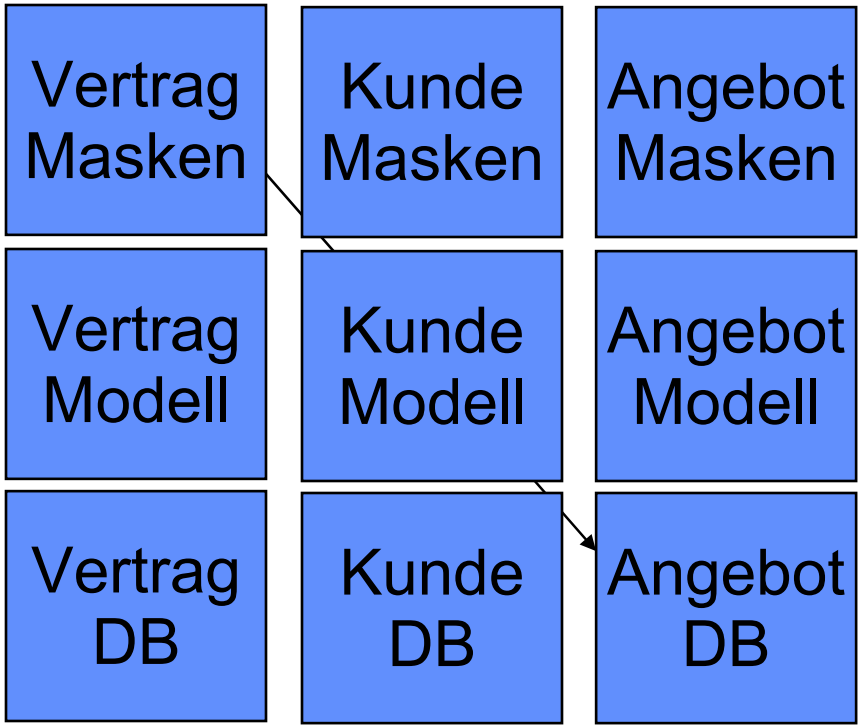
Schneiden von Komponenten

Technische Sicht auf ein System



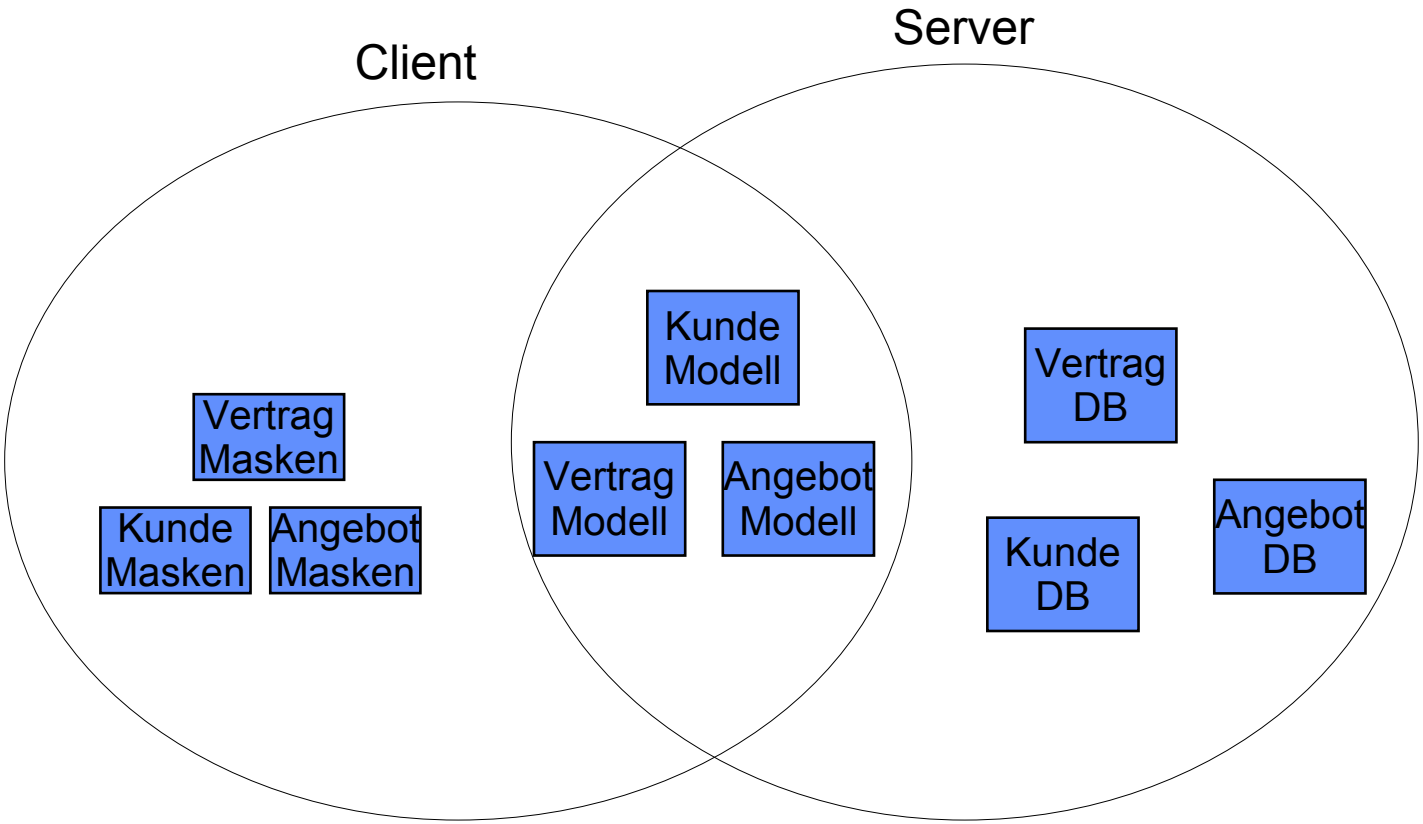
Schneiden von Komponenten

Fachliche Sicht ergänzt technisches Modell



Schneiden von Komponenten

Deployment Sicht



Inhalt

- Motivation
- Architektur
- **Architektur Evolution**
- OSGi
- Refactoring Beispiel
- Migration bestehender Systeme
- Probleme mit OSGi
- Fazit

Änderungen an der Architektur

- Komponenten
  - Austausch von Komponenten
  - Hinzufügen von Komponenten
  - Entfernen von Komponenten
- Beziehungen
  - Beziehungen ändern Richtungen
  - Schnittstelle der Beziehung ändert sich
- Rahmenbedingungen ändern sich

Beispiele für Strukturveränderungen

- Single- vs. Multiuser
- Multicore Prozessoren
- Objektorientiert- vs. Relational-DB
- Web vs. Rich Client
- Inversion of Control (Dependency Injection)

Anforderungen an evolutionäre Architektur

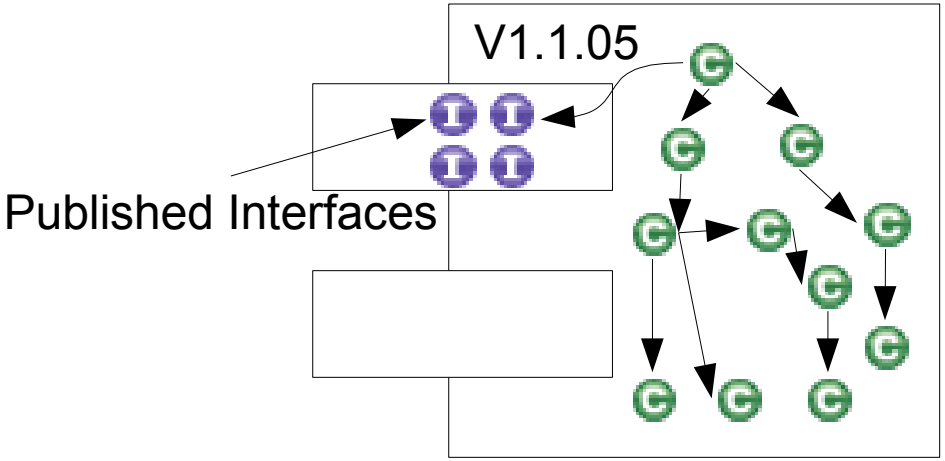
- Wiederverwendbare Komponenten
- Isolation von Komponenten
- Definierte Abhängigkeiten
- Änderungen mit möglichst wenig Auswirkung auf andere Komponenten
- Leichte Austauschbarkeit von Komponenten
- Unterstützung von Published Interfaces

Published Interfaces

Artikel: Public versus Published Interfaces

von Martin Fowler

http://martinfowler.com/ieeeSoftware/publishe



Packages reichen nicht für Komponenten, somit Public auch nicht.

Inhalt

- Motivation
- Architektur
- Architektur Evolution
- **OSGi**
- Refactoring Beispiel
- Migration bestehender Systeme
- Probleme mit OSGi
- Fazit

Ein Architekturmuster: OSGi

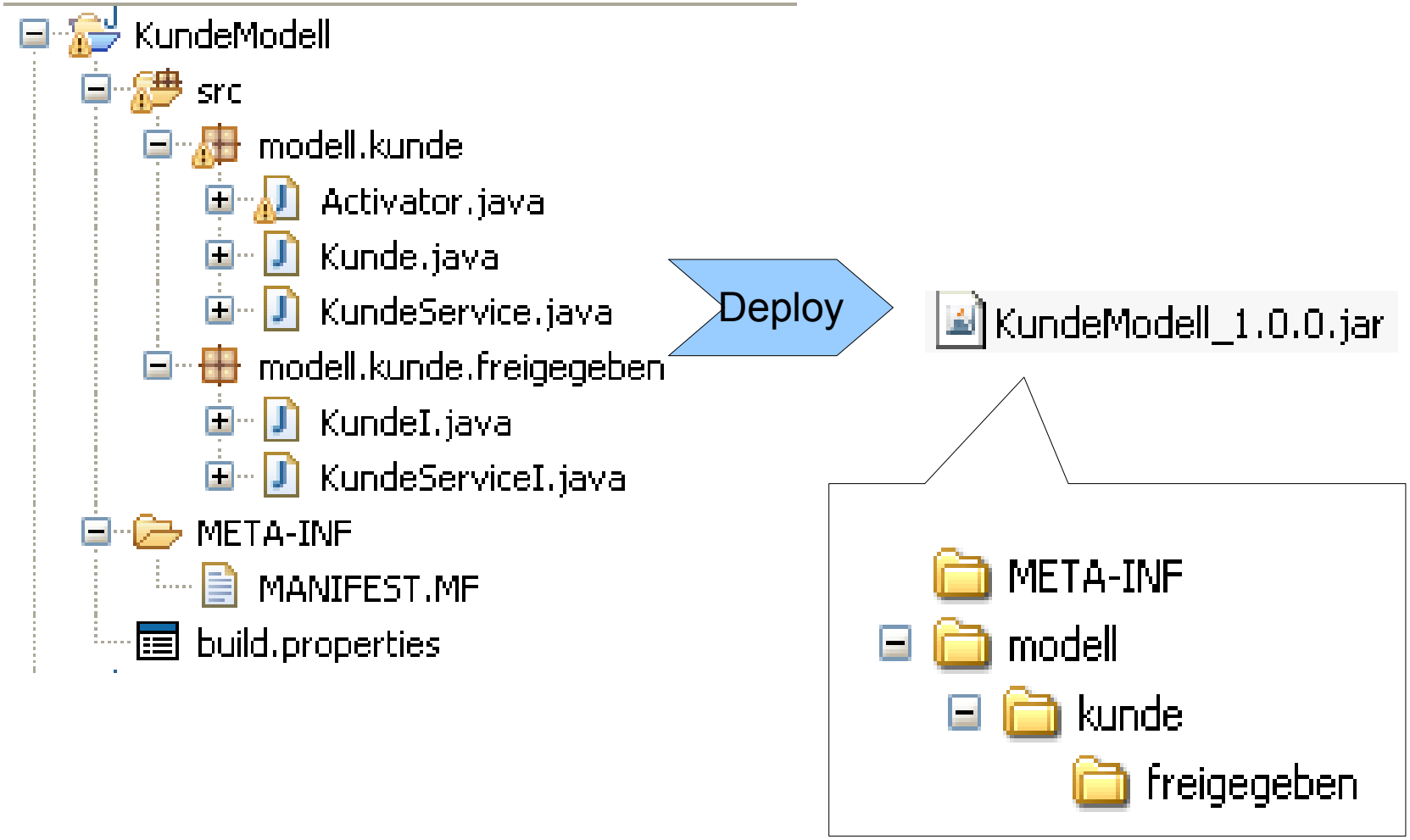
Open Service Gateway Initiative

- gegründet 1999
- Spezifikation eines Komponenten-Containers in Java
- Ursprung embedded Bereich => geringer Overhead
- Verschiedene Implementierungen
- Verwendung im Enterprise Umfeld

OSGi Spezifikation im Detail

- Bundles (Komponenten)
- Dependency Management
- Classloader pro Bundle
- Published Interfaces
- Services

Ein Bundle in OSGi

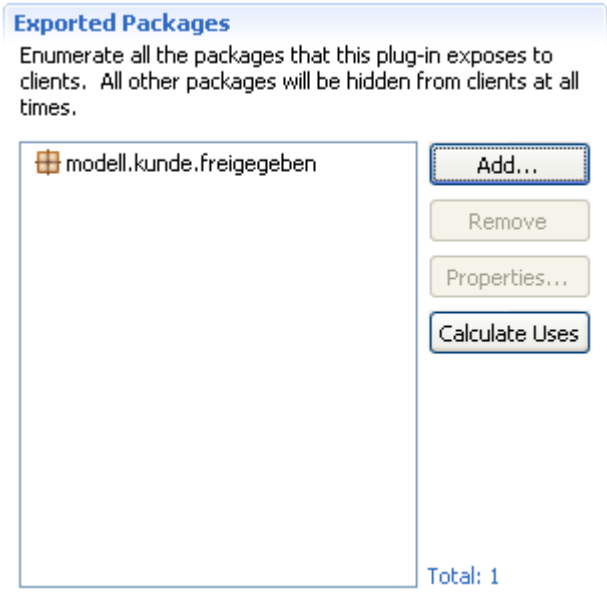


Manifest

```
Manifest-Version: 1.0
Bundle-ManifestVersion: 2
Bundle-Name: KundeModell Bundle
Bundle-SymbolicName: KundeModell
Bundle-Version: 1.0.0
Bundle-Activator: modell.kunde.Activator
Import-Package: org.osgi.framework;version="1.3.0",
 org.osgi.util.tracker;version="1.3.1"
Export-Package: modell.kunde.freigegeben
```

Published Interfaces

- Schnittstellen eines Bundles
- Zusätzliche Sichtbarkeitsebene
- Compile- / Laufzeit

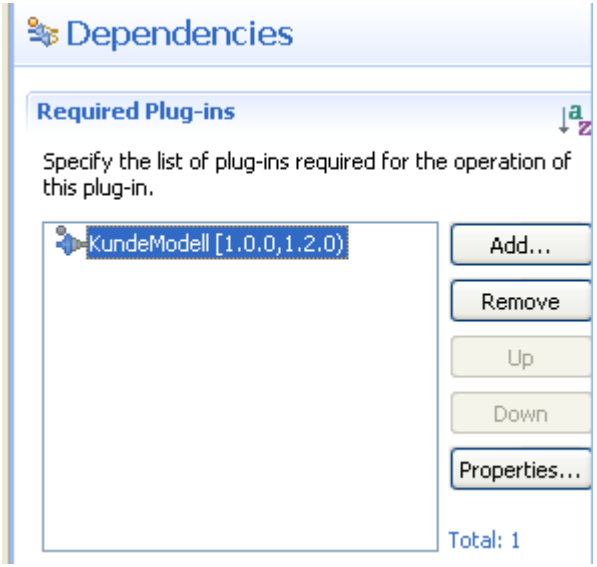



```
Manifest-Version: 1.0
Bundle-ManifestVersion: 2
Bundle-Name: KundeModell Bundle
Bundle-SymbolicName: KundeModell
Bundle-Version: 1.0.0
Bundle-Activator: modell.kunde.Activator
Import-Package:
org.osgi.framework;version="1.3.0",
 org.osgi.util.tracker;version="1.3.1"
Export-Package: modell.kunde.freigegeben
```

Abhängigkeit (1)

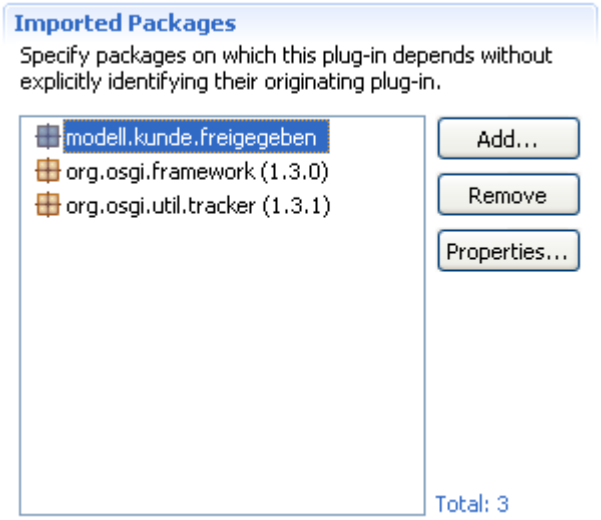
- Nur Published Interfaces sichtbar für andere Bundles
- Jedes Bundle ein Classloader
- Abhängigkeiten werden beschrieben durch
  - Required Bundles
  - Imported Packages

Abhängigkeit (2)




```
Manifest-Version: 1.0
Bundle-ManifestVersion: 2
Bundle-Name: KundeView Bundle
Bundle-SymbolicName: KundeView
Bundle-Version: 1.0.0
Bundle-Activator: view.kunde.Activator
Import-Package: org.osgi.framework;version="1.3.0",
 org.osgi.util.tracker;version="1.3.1"
Require-Bundle: KundeModell;bundle-version="[1.0.0,1.2.0)"
```

Abhängigkeit (3)

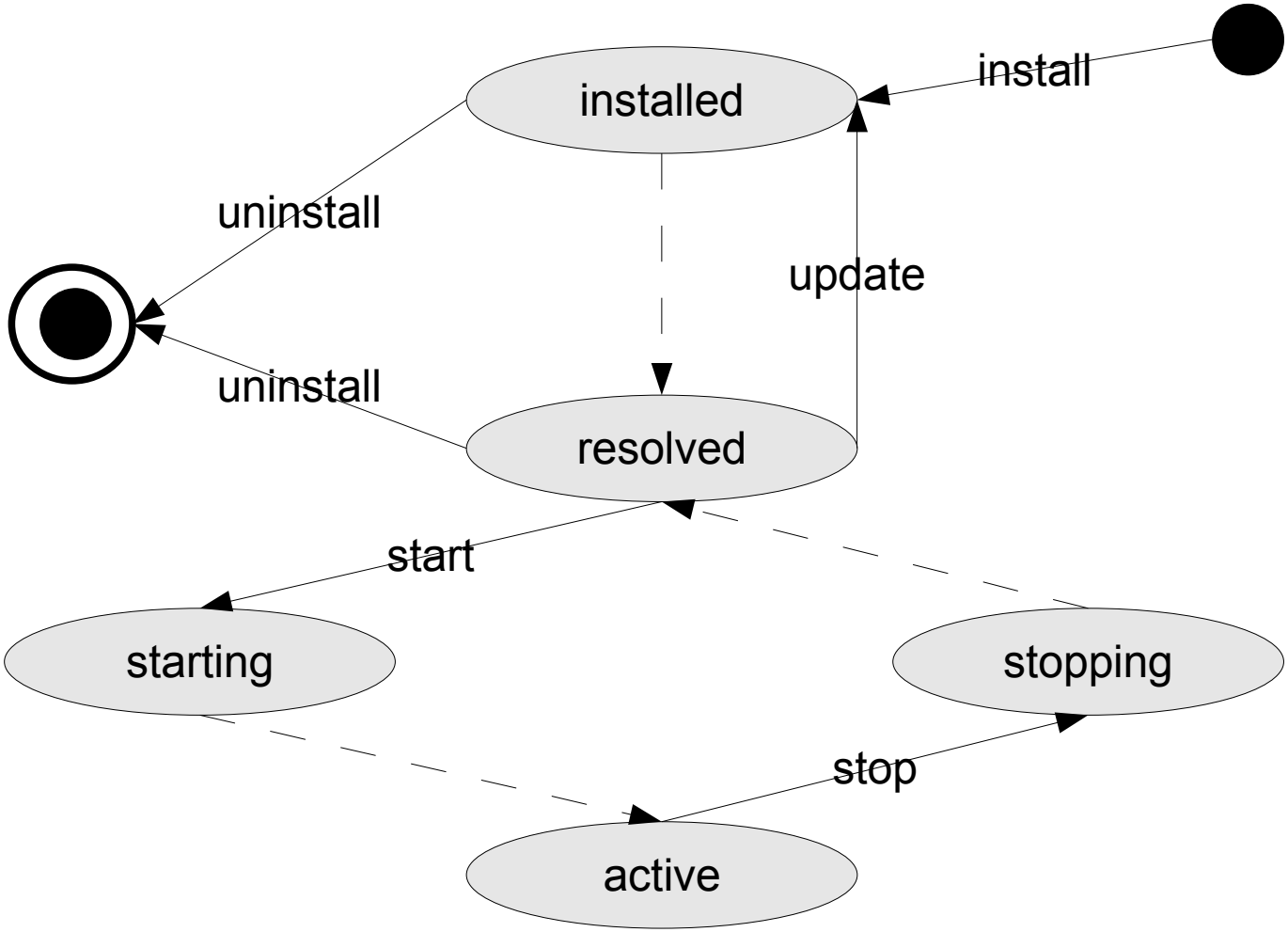



```
Manifest-Version: 1.0
Bundle-ManifestVersion: 2
Bundle-Name: KundeView Bundle
Bundle-SymbolicName: KundeView
Bundle-Version: 1.0.0
Bundle-Activator: view.kunde.Activator
Import-Package:
modell.kunde.freigegeben;version="[1.0.0,1.2.0)",
 org.osgi.framework;version="1.3.0",
 org.osgi.util.tracker;version="1.3.1"
```

Activator (Singleton / einer pro Bundle)

```
public class Activator implements BundleActivator {

  private static Activator activator=null;

  public void start(BundleContext context) throws Exception {
    activator=this;
  }

  public void stop(BundleContext context) throws Exception {
    activator=null;
  }

  public static Activator getActivator() {
    return activator;
  }


}
```

Zustände eines Bundles



— —► Automatisch

Service Nutzen

```
...
private ServiceTracker kunderServiceTracker;
...

public void start(BundleContext context) throws Exception {
    ...
    kunderServiceTracker = new ServiceTracker(context,
    KundeServiceI.class.getName(), null);
    kunderServiceTracker.open();
}

public KundeServiceI getKundeService(){
  return (KundeServiceI) kunderServiceTracker.getService();
}
```

Service Anbieten

```
private ServiceRegistration serviceRegistration;

public void start(BundleContext context) throws Exception {


   KundeServiceI kundeServiceI= new KundeService();

   serviceRegistration =
context.registerService(KundeServiceI.class.getName(),
                        kundeServiceI,
                           new Hashtable());
}

public void stop(BundleContext context) throws Exception {
               serviceRegistration.unregister();
       }
```

Inhalt

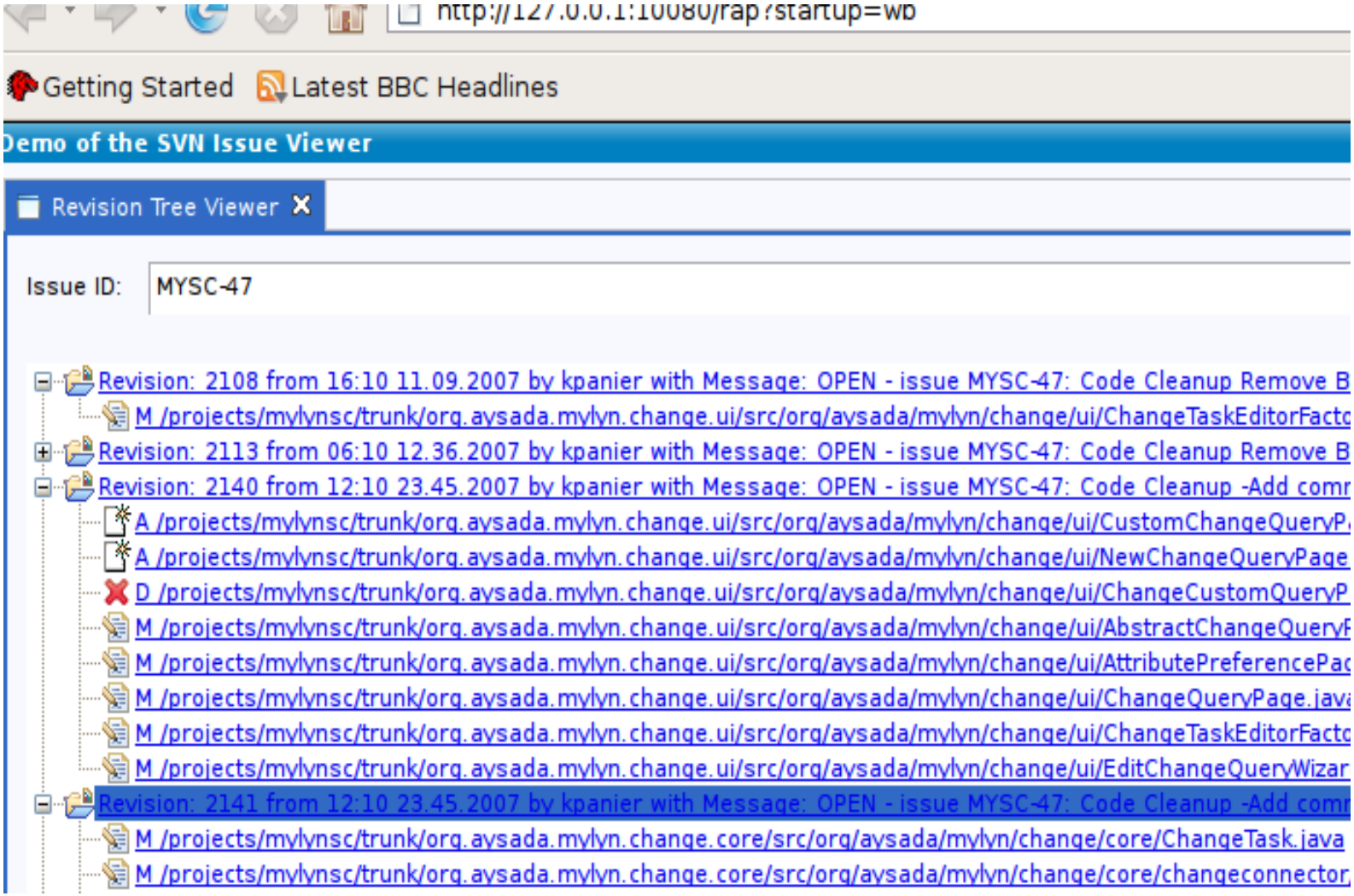
- Motivation
- Architektur
- Architektur Evolution
- OSGi
- **Refactoring Beispiel**
- Migration bestehender Systeme
- Probleme mit OSGi
- Fazit

Beispiel Architektur Refactoring mit OSGi

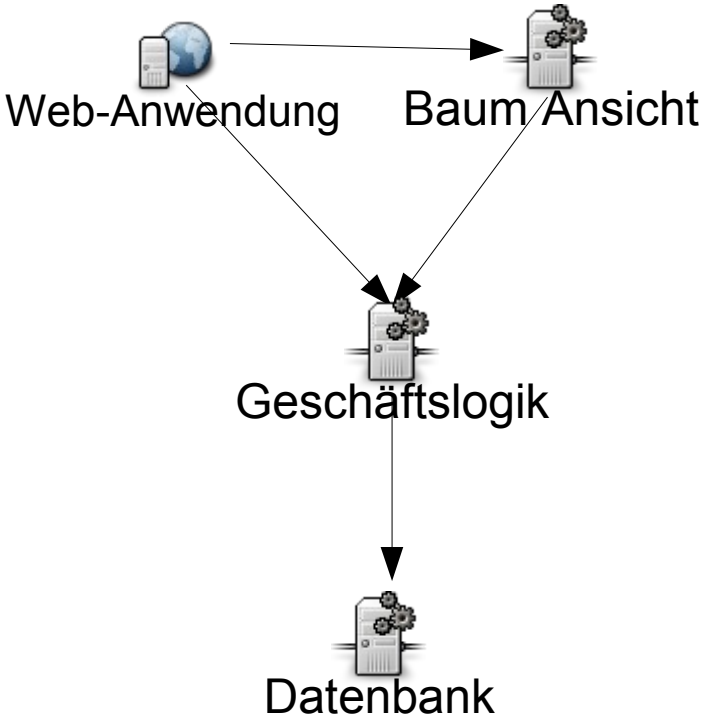
Frage:

Welche Dateien wurden für Bug MYSC-47 geändert?

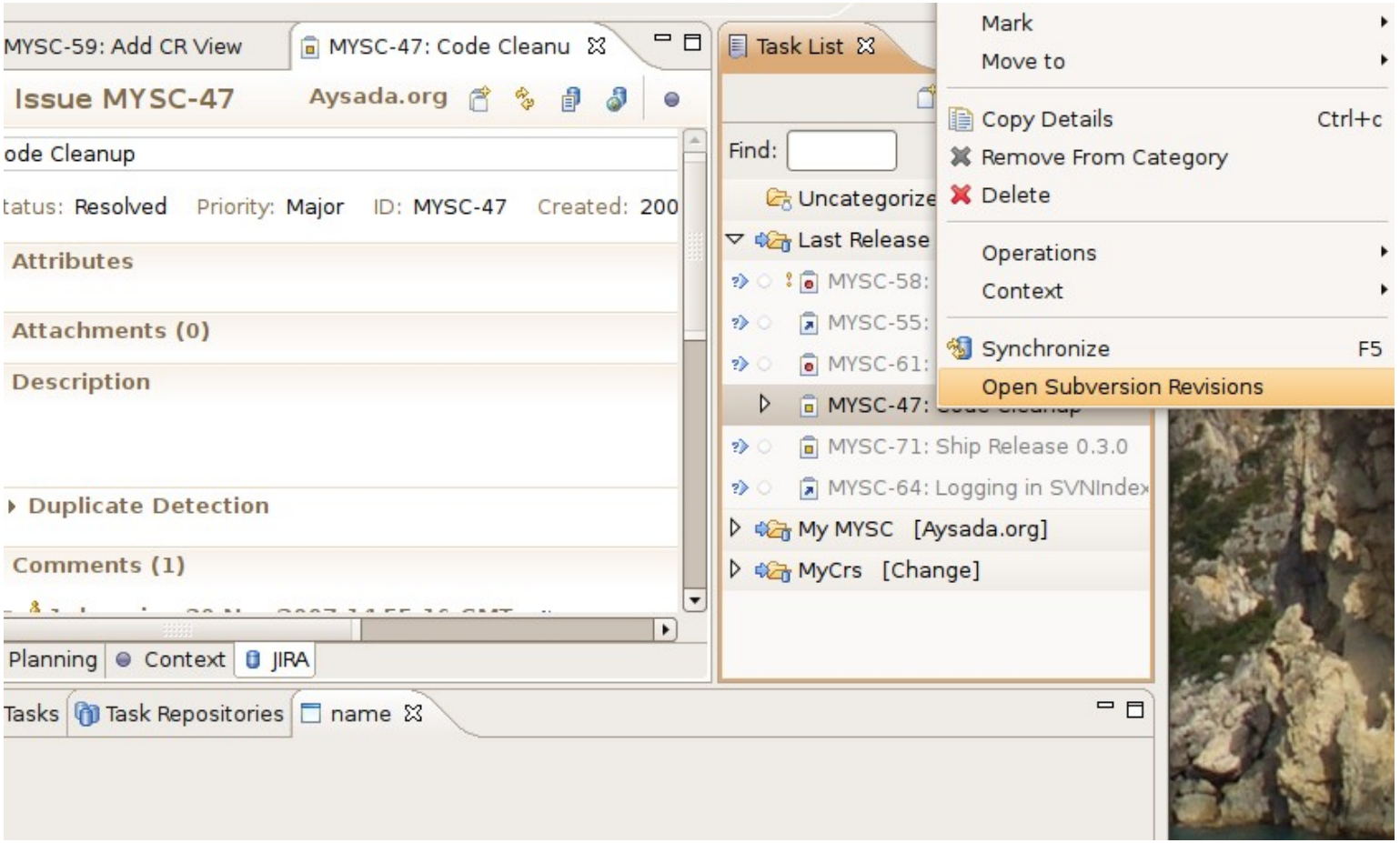
WEB-Anwendung mit der Antwort



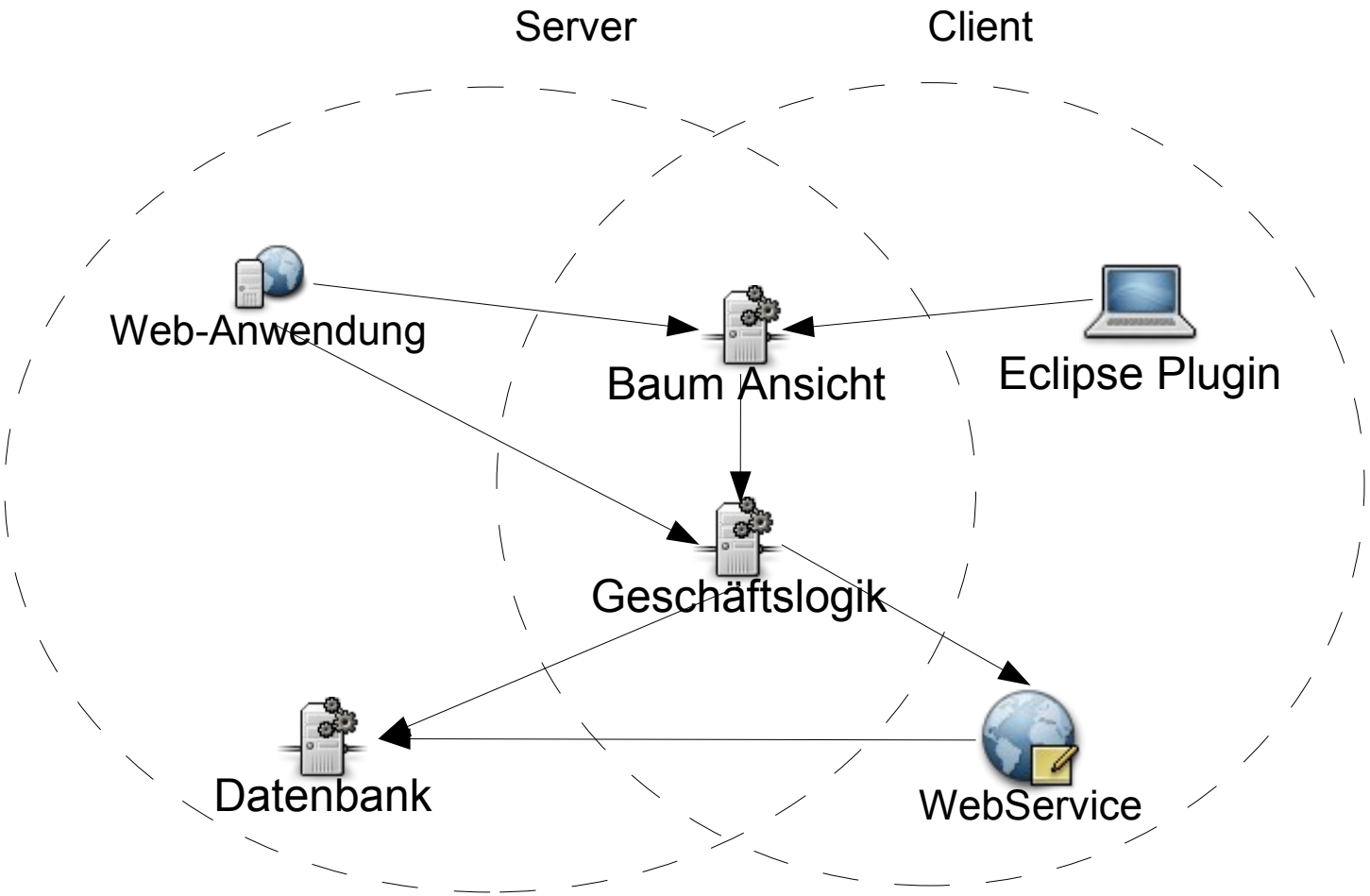
Ausgangsarchitektur



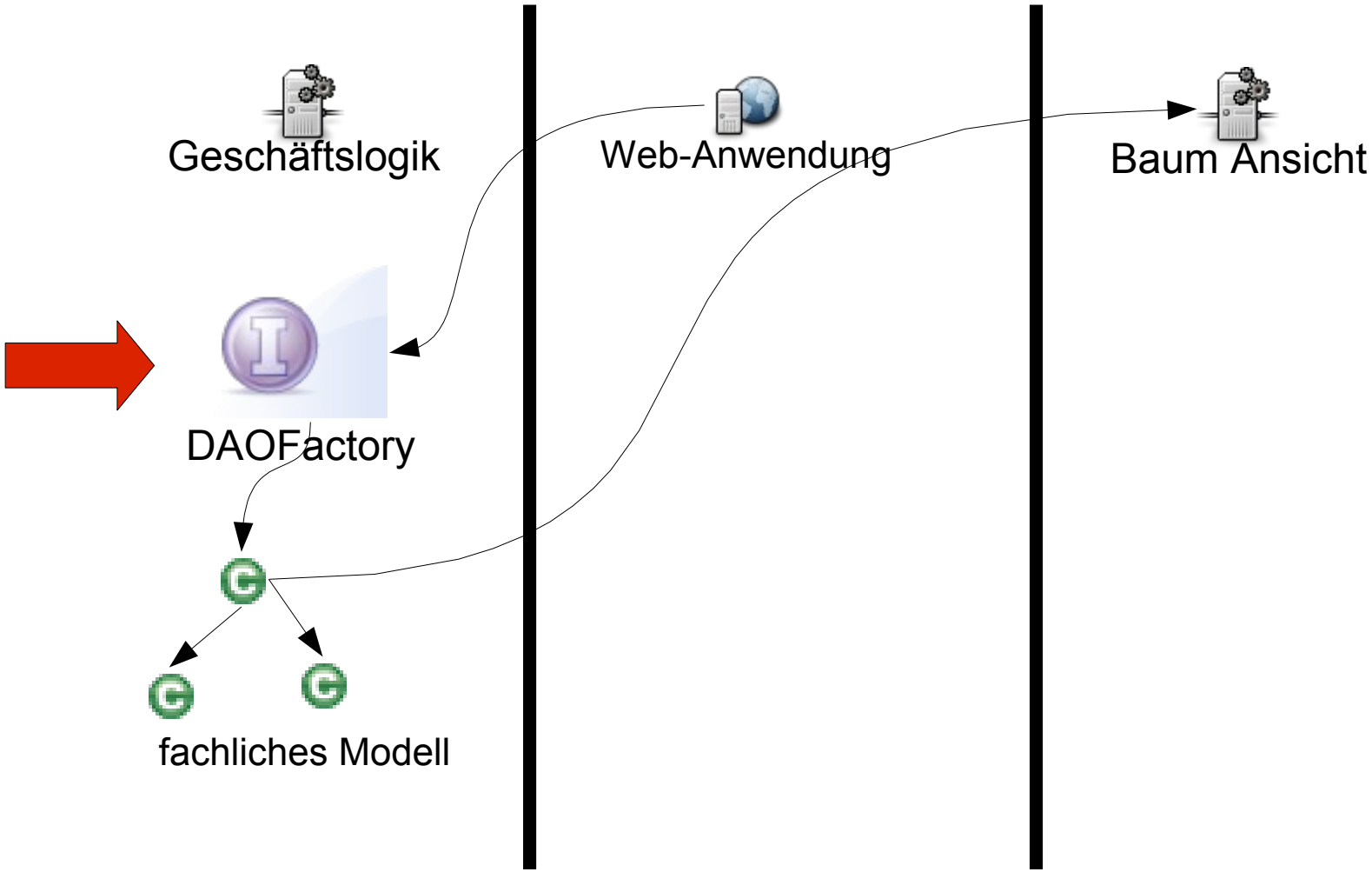
Neue Anforderung: Baumanchsicht auch in Eclipse



Ziel Architektur



Wo die Architektur ändern?



Liste der Tasks

- MYSC-66 DAO als Service konsumieren
- MYSC-67 Geschäftslogik Bundle aufteilen
- MYSC-68 Webservice DAO implementieren
  - MYSC-73 Jedes DAO ein eigener OSGi Service
  - MYSC-74 Fachliches Modell für Webservice vorbereiten
- MYSC-69 Baum Ansicht als Eclipse View implementieren

MYSC-66 DAO als Service konsumieren

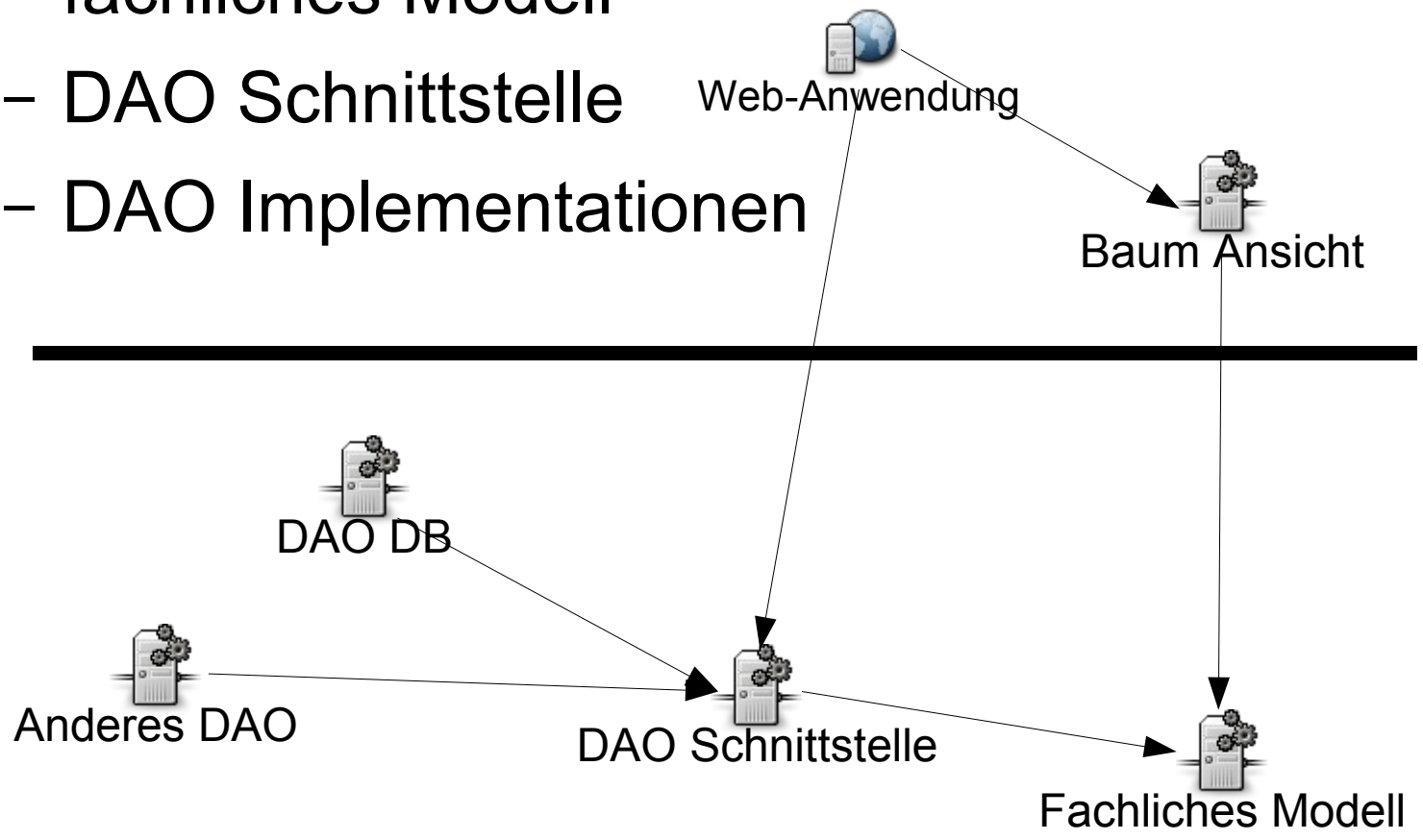
- Refactor DAO's: Extract Interface
- Implemenation in internen Packages verstecken
- Interface Implementation als OSGi-Service konsumieren
  - Activator der Geschäftslogik registriert eine Implementation des IDAOService Interface.
  - UI Activator holt über den Service eine Implementation des Interfaces

Fazit MYSC-66

- Trennung zwischen Interface und Implementation. Nur das Interface Package wird exportiert.
- Die vorherige Factory Klasse wird nun als Service angesprochen. Der Service kann daher zur Laufzeit ausgetauscht werden.
  - ServiceTracker handelt Referenzen auf Services

MYSC-67 Geschäftslogik Bundle aufteilen

- Aufspalten des „großen“ Bundles.
  - fachliches Modell
  - DAO Schnittstelle
  - DAO Implementationen

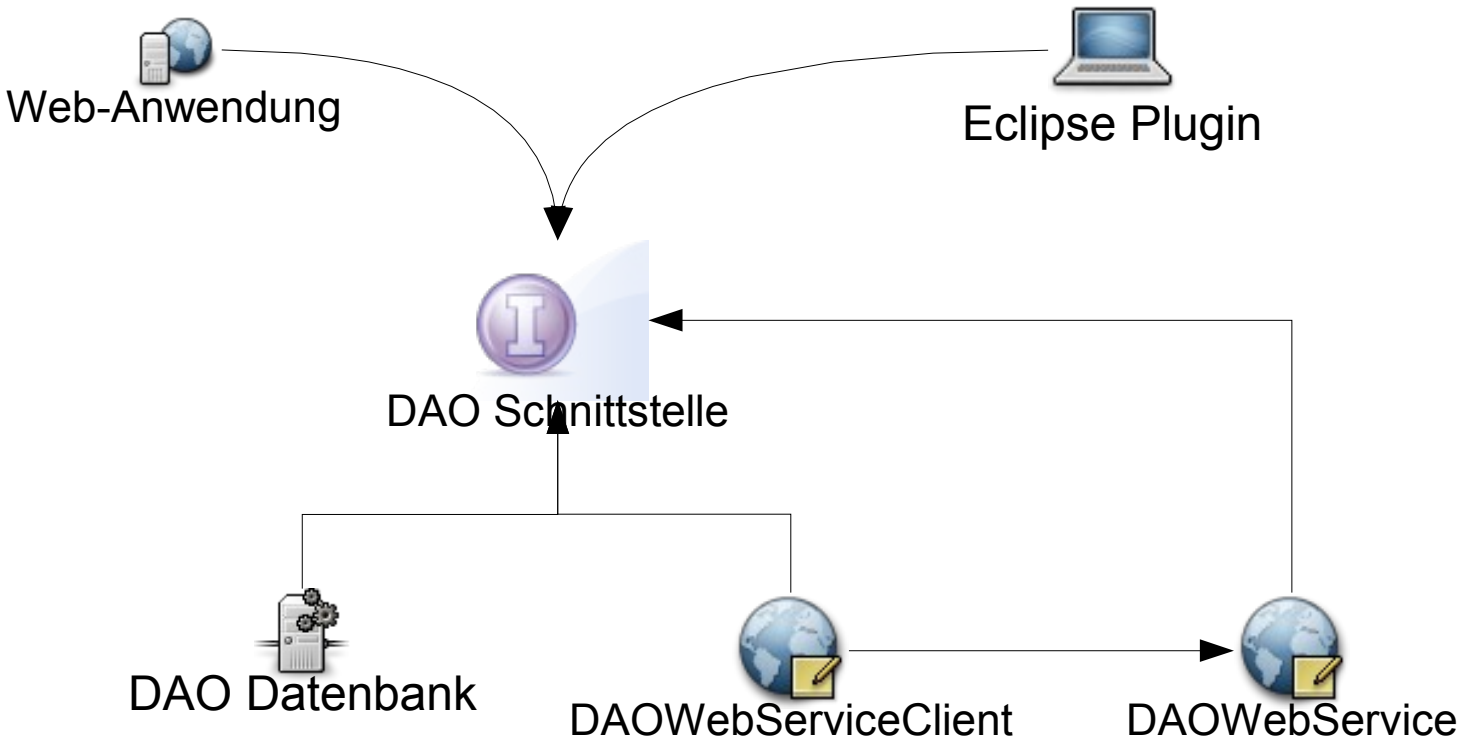


Fazit MYSC-67

- Verstecken der Implementation in eigenen Bundle
- Zyklen zwischen den Bundles müssen aufgelöst werden
  - Vergessene Architekturverletzungen werden gefunden

```
/* Replace Method with dependency Injection */
private void loadModifications() {
        modifications = Activator.getDefault().getDaoService().
getChangesPathDAO().loadModificationsFor(getRevisionNumber());
}
```

MYSC-68 Webservice DAO implementieren



Bei der Implementierung fällt auf:

- Jedes DAO sollte als Service verfügbar sein.
  - Sonst müsste eine aufwendige Webservice Factory entworfen werden.
  - Es würden immer alle DAOs über die API bereitgestellt werden
- Fachliches Modell eignet sich noch nicht für den Transport über SOAP

Fazit MYSC-73 OSGi Services anpassen

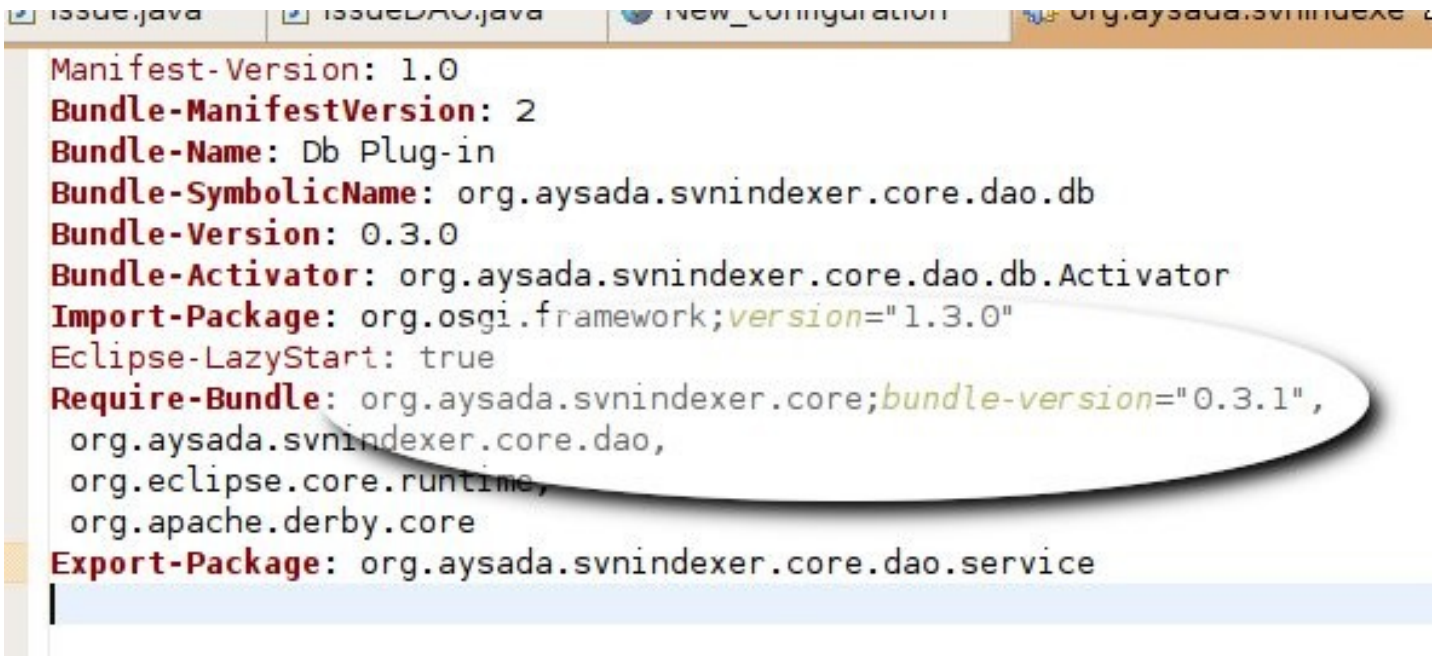
- Jedes DAO als eigener Service.
- Factory wird Connectionpool der DB
- Einführen des DAOServiceLookUpUtil's im Schnittstellen Bundle.

→ Architektur fühlt sich besser an.

  - DAO sind weniger gekoppelt.
  - Verantwortungen klarer
  - Coderedundazen verringert

Fazit MYSC-74 Fachliches Modell anpassen

- Änderung an API Klassen
  - Einführung einer neuen Bundle Version
  - Abhängigkeiten der Client Bundles



Fazit (I) MYSC-68 Webservice

- Einführen eines neuen Bundles, das DAO Zugriff als Webservice implementiert
  - Webservice ist abhängig vom fachlichen Modell Version 0.3.1
- Webservices in OSGi bereitstellen
  - OSGi ist kein Servletcontainer.
  - Servlets müssen selbst registriert werden.
  - Web-Inf im Bundle, dass das Servlet registriert.

Fazit (II) MYSC-68 Webservice

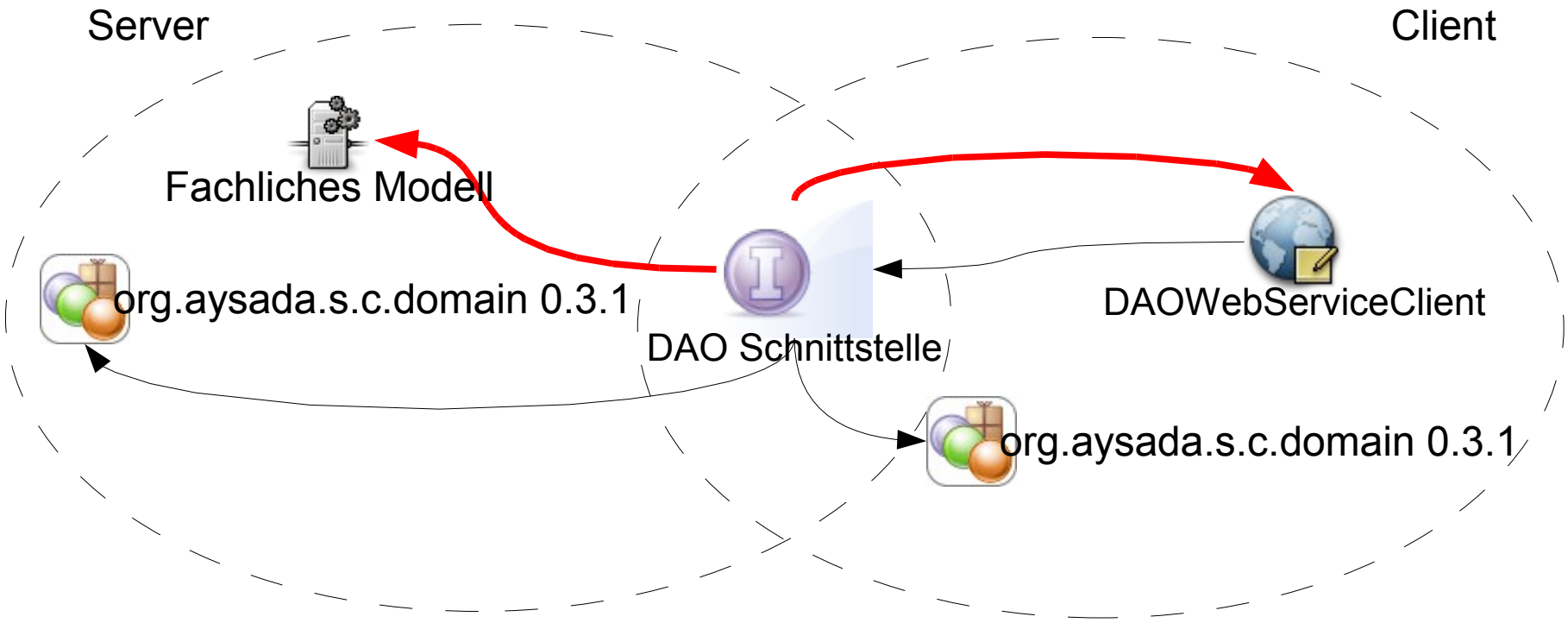
- Wie neue Biblotheken einführen:
  - jeweils als eigenes Bundle
    - am besten aus zentralen Repositories
      - Eclipse Orbit
      - Felix
  - ins Webservice Bundle mit aufnehmen
  - ein Bundle für voneinander stark abhängige Bibliotheken

MYSC-69 Hinzufügen des Views in die Eclipse IDE

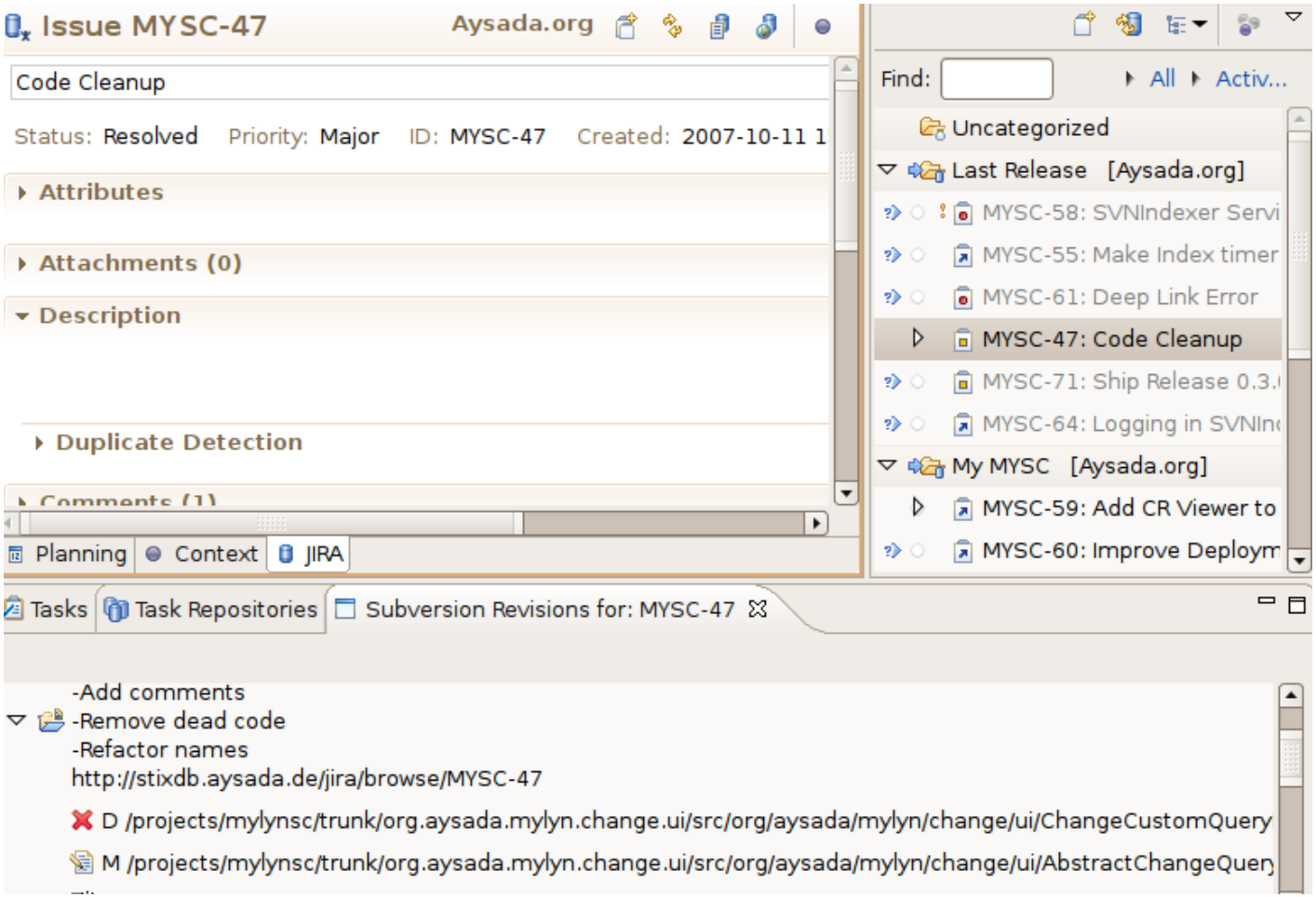
- WS Client generieren als neues Bundle
- DAO Schnittstellen Bundle im Client installieren
- Variante vom View entwickeln
  - Bundle übernehmen scheitert an Bundle Abhängigkeiten
- Client implementiert DAO Interface und registriert den OSGi-Service

Fazit MYSC-69

- DAO Inferface Bundle:
  Abhängigkeit von Bundle auf Package umstellen.



Fertig


Issue MYSC-47
Aysada.org
Code Cleanup
Status: Resolved
Priority: Major
ID: MYSC-47
Created: 2007-10-11 1
Attributes
Attachments (0)
Description
Duplicate Detection
Comments (1)
Planning
Context
JIRA
Find:
All
Activ...
Uncategorized
Last Release [Aysada.org]
MYSC-58: SVNIndexer Servi
MYSC-55: Make Index timer
MYSC-61: Deep Link Error
MYSC-47: Code Cleanup
MYSC-71: Ship Release 0.3.
MYSC-64: Logging in SVNIn
My MYSC [Aysada.org]
MYSC-59: Add CR Viewer to
MYSC-60: Improve Deploym
Tasks
Task Repositories
Subversion Revisions for: MYSC-47
-Add comments
-Remove dead code
-Refactor names
http://stixdb.aysada.de/jira/browse/MYSC-47
D /projects/mylynsc/trunk/org.aysada.mylyn.change.ui/src/org/aysada/mylyn/change/ui/ChangeCustomQuery
M /projects/mylynsc/trunk/org.aysada.mylyn.change.ui/src/org/aysada/mylyn/change/ui/AbstractChangeQuery

Beispiel

- Anforderung
  http://jira.aysada.org/jira/browse/MYSC-59
- Quellcode
  Login: guest/guest
  https://www.aysada.org/svnrep/develop/projects/svnindexer/branches/MYSC-59/
  https://www.aysada.org/svnrep/develop/projects/mylynsc/branches/MYSC-59/

Inhalt

- Motivation
- Architektur
- Architektur Evolution
- OSGi
- Refactoring Beispiel
- **Migration bestehender Systeme**
- Probleme mit OSGi
- Fazit

Migration zu OSGi

- Eigene Bibliotheken / Java Projekte als Bundles
  - Klare Abhängigkeiten und keine Zyklen
  - Versionierte Jars, die auch außerhalb OSGi laufen
- Einführung von Published Interfaces
- Services benutzen
  - Bundles lassen sich austauschen (Hotfixes)
  - Jetzt wird ein OSGi-Container zwingend.

Probleme OSGi

- Nur java.* Packages vom JRE sichtbar.
- Classloaderprobleme bei Reflection
- Automatischer Build benötigt anderes Tooling
  - Eclipse PDE
  - Maven Plug-Ins (pom.xml oder manifest.mf)
- Keine Gruppierung von Bundles
- Zustände von Objekten in ausgetauschten Bundles

Abgrenzung zur Java Spezifikation

OSGi ist nicht wie JEE Bestandteil der Java Spezifikation

- JSR 208 Java Business Integration
- JSR 277 Komponenten für Java
- JSR 294 Superpackages

Fazit

- OSGi strukturiert Systeme und unterstützt sich ändernde Architekturen
- OSGi hat geringen Overhead, leichte Einführung
- Classloader Baum zur Compile- und Laufzeit sind gleich
- Nicht einfach im JEE Umfeld einsetzbar

Ausblick

- Bessere Build Unterstützung
- Bundle Repositories
- Bessere Integration
  - Application Server (Websphere, JBoss, BEA)
  - Swordfish
  - Spring
- Verteilte OSGi Umgebungen
- Universal OSGi: andere Sprachen

Fragen ?

Literatur

- Links
  - http://www.osgi.org
  - http://www.eclipse.org/equinox
  - http://www.osoa.org/download/attachments/250/Powe
  - Links: http://martinfowler.com/ieeeSoftware/published.pdf
  - http://www.eclipse.org/corona/doc/corona-tptp-muse-eclipseWorld06-2006%2008%2004-GA.pdf
  - http://javamagazin.de/mediapool/podcasts/java/jax_link/jax_link_PAF4.pdf
  - http://qurl.com/tdvnp