Orientation in Objects

OSGi in fremden Modulwelten

Orientation in Objects GmbH

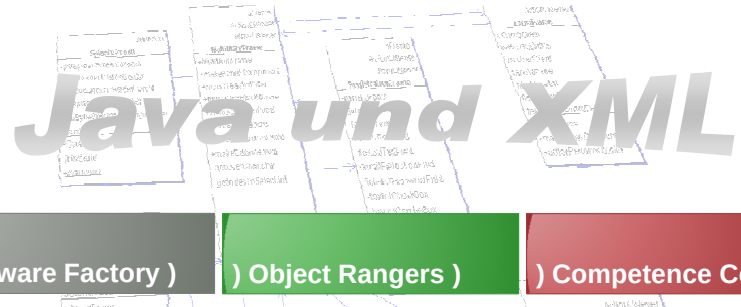
Weinheimer Str. 68
68309 Mannheim

www.oio.de
info@oio.de

Version: 1.1

Java, XML und Open Source seit 1998

Orientation in Objects



) Software Factory)	) Object Rangers)	) Competence Center)
• Schlüsselfertige Realisierung von Java Software	• Unterstützung laufender Java Projekte	• Schulungen, Coaching, Weiterbildungsberatung, Train & Solve-Programme
• Individualsoftware • Pilot- und Migrationsprojekte • Sanierung von Software • Software Wartung	• Perfect Match • Rent-a-team • Coaching on the project • Inhouse Outsourcing	• Methoden, Standards und Tools für die Entwicklung von offenen, unternehmens-weiten Systemen

© 2010 Orientation in Objects GmbH

Migrationsprojekte

Orientation in Objects

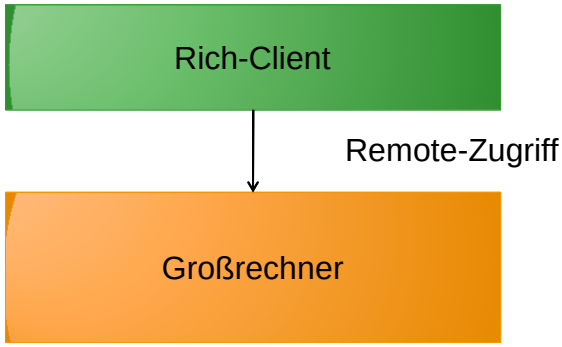
- 1999 BS2000 → C++ Server/CORBA/XML-RPC/Java-Client
  - Aufwand: min. 200 Dev/a
  - Ergebnis: abgebrochen nach mehr als 3 Jahren Laufzeit
- 2003 ADABAS/Natural → J2EE+ Java Rich Client
  - Aufwand: ca. 5 Dev/a
  - Ergebnis: Pilotierung , Go Live Plangemäß Q 2/2004
  - ca. 400 Terminals wurden demontiert ;)
- 2008 RPG?→ Java

Ausgangssituation

Orientation in Objects

- Bestehende Großrechner-Anwendung
  - Mehr als 20 Jahre Produktionserfahrung
  - Mehr als 20 Kundeninstallationen
  - Sehr kosteneffizienter Betrieb (near zero administration)
  - ASP möglich
- Zugriff über Terminal-Clients
  - Bis zu 4-stellige Benutzerzahlen pro Installation
  - Effektive praxiserprobte Benutzerführung
- Große Menge Business-Logik, die nicht mal schnell in Java reproduziert werden kann
  - >100 Personenjahre Entwicklungsaufwand
  - >10^6 lines of code

Das ist doch einfach!?!



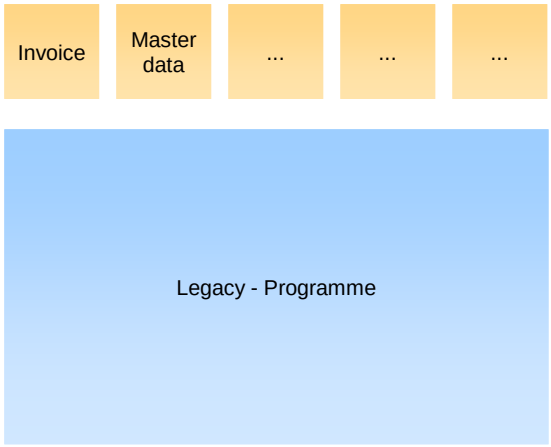
Gliederung

- **Problemstellung**
- Lösungsansätze
- Praktische Umsetzung

- **Migration**
- Integration
- Update
- Qualitätsmanagement

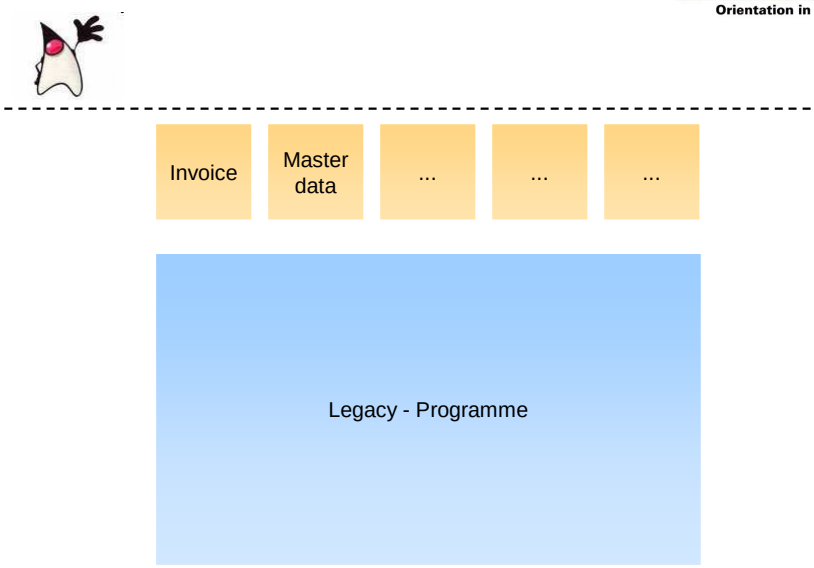
Logischer Modulaufbau des Produktes

Orientation in Objects



Migrationsvision nach Java (1)

Orientation in Objects



Migrationsvision nach Java(2)

Invoice

Master data | ... | ... | ...

Legacy - Programme

Migrationsvision nach Java(3)

Invoice | Master data

... | ... | ...

Legacy - Programme

Migrationsvision nach Java(4)

Invoice | Master data | ... | ... | ...

Legacy - Programme

Gliederung

- **Problemstellung**
- Lösungsansätze
- Praktische Umsetzung

- Migration
- **Integration**
- Update
- Qualitätsmanagement

Modularisierung...

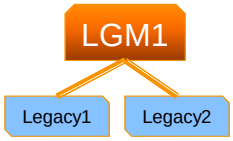
Orientation in Objects

- Ein Meer von Definitionen im Stil von
  "A module is a self-contained component of a system…"
- oft an Randbedingungen gekoppelt
  - z. B. logische Ebene
    - **Abstraktionslevel**
    - **Programmiersprache**
  - z.B. physisch/technische Ebene
    - **Repräsentation**
    - **Transport**
    - **Persistenz**
- Was passiert beim zwangsweisen Kontakt von "Modulwelten"

Legacy Module (LGM)

Orientation in Objects

- Vergleichbar mit Dokumentenbasierten Webservices
  - Also keine direkte Abbildung Serviceschnittstelle (mit mehrere Methoden) auf Legacy Programm möglich
  - Eine Schnittstelle im „Java" Sinn würde immer auf mehrere Legacy Programme zeigen
  - Hinter LGMs verstecken sich Legacy Programme
  - LGMs werden speziell für die Java Anwendung bereitgestellt und werden von keinem anderem verwendet



Gliederung

Orientation in Objects

- **Problemstellung**
- Lösungsansätze
- Praktische Umsetzung

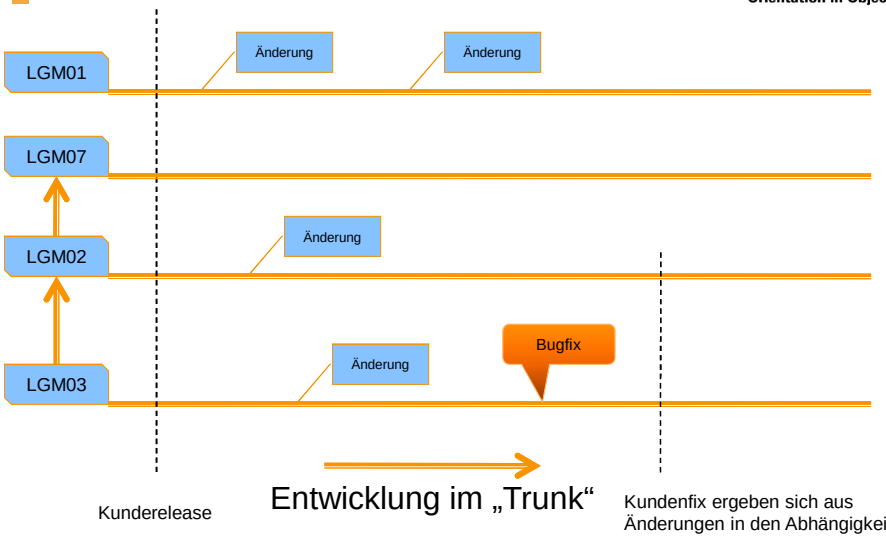
- Migration
- Integration
- **Update**
- Qualitätsmanagement

Update-Strategie

Orientation in Objects

- Legacy (Ist)
  - Update einzelner Großrechner-Programme
  - Update von abhängigen Programmen
  - Pflege der Abhängigkeiten durch Entwickler
- Java (Soll)
  - Modul-weise Updates
  - Inkrementelle Client Updates wegen Netzwerk-Last
  - Synchrone Updates Großrechner & Rich Client
  - Hotfix-Fähigkeit

Updatestrategie des Legacy-Produktes



Gliederung

- **Problemstellung**
- Lösungsansätze
- Praktische Umsetzung

- Migration
- Integration
- Update
- **Qualitätsmanagement**

Probleme bei der Implementierung des Konzeptes

Orientation in Objects

- Reproduzierbarkeit von Builds sicherstellen
  - Java Module sind kaum ohne LGM lauffähig
- Versionen von Großrechner-Services und Client-Artefakten zuordnen
  - Freigabe Prozess
  - Bereitstellung von Referenzdaten
- Issuetracking
  - Zuordnung von Issues Versionsständen von Kunden
  - Zuordnung von Issues innerhalb der Architektur
  - Ableitung Updatestrategien zum Bugfixing
- Customizing

- Randbedingung: QM-Strategie im Legacy Produkt ist fix

Gliederung

Orientation in Objects

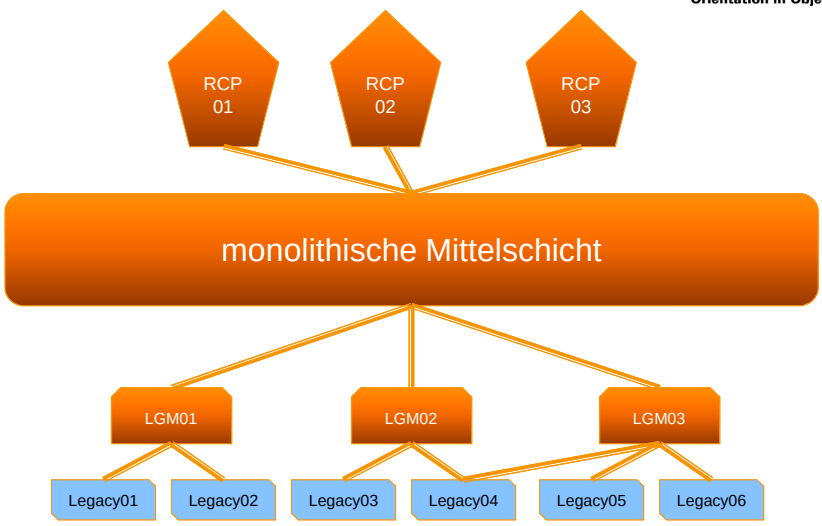
- Problemstellung
- **Lösungsansätze**
- Praktische Umsetzung

- **Service Granularität**
- Dependency Management
- Silokonzept

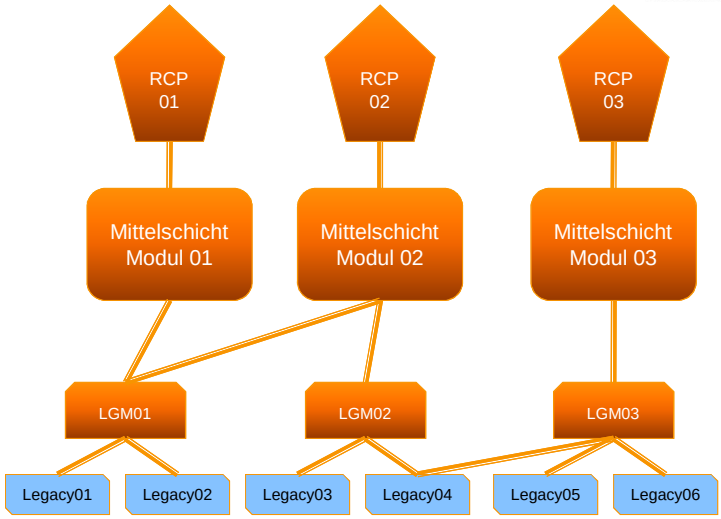
AS/400 Connector

- JTOpen
  - Formerly known as „IBM Toolbox for Java“
- Zugriffe auf
  - **Database**
  - **JDBC (SQL) and record-level access (DDM)**
  - **Integrated File System Program calls (RPG, COBOL, service programs, etc)**
  - **Commands Data queues**
  - **Data areas**
  - **Print/spool resources**
  - **Product and PTF information**
  - **Jobs and job logs Messages, message queues, message files**
  - **Users and groups User spaces**
  - **System values**
  - **System status**
- http://jt400.sourceforge.net/

Java Anbindung Variante 1: monolithische Mittelschicht



Variante 2: modulare Mittelschicht



Schnittstelle zwischen Legacy und Java definieren

Java

Schnittstellendefinition
- Infrastruktur
- Operationen
- Datenstrukturen (DOM)
- Metadaten

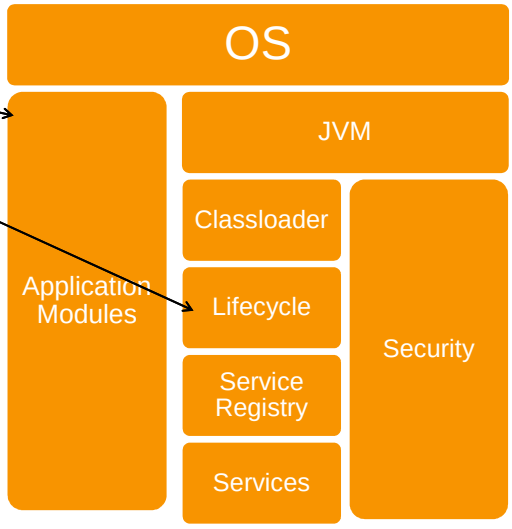
Legacy

Service-Schnittstellen

- Statt einzelne LGMs anzusprechen jetzt Services ansprechen
- Services werden gemeinsam (Lieferant und Kunde) definiert
  - Service-Anbindung durch Lieferant
  - Service-Implementierung durch Kunde
- Abhängigkeiten werden im Legacy System verwaltet

Benutzte OSGi - Layers

- Component Definition
- Component Management



Gliederung

- Problemstellung
- **Lösungsansätze**
- Praktische Umsetzung

- Service Granularität
- **Dependency Management**
- Silokonzept

Probleme

- Modularisierung
  - Größere Komponenten
  - Kleiner als Deployment Artefakt
- Integration
- Abhängigkeiten zwischen Module
  - Definieren
  - Compiletime
  - Laufzeitverhalten
- Versionierung

Die Lösungsansätze

- Wir lösen es in der Build-Infrastruktur und haben zur Laufzeit ein Monolith...
  - Schon mal mit Maven, Ivy & Co. gelöst.

oder...

- OSGi !!!!

Java – Legacy connection dependencies?

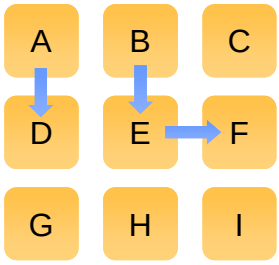
Orientation in Objects

Java Mittelschicht Legacy-Anbindung

Orientation in Objects

- Führt zu Abhängigkeitsgraph „außerhalb“ des Legacy Systems
- Je nach Ausführung werden unter Umständen mehr oder weniger Abhängigkeiten gebildet:
  - Eine monolithische Mittelschicht bindet alle verwendete Legacy Module zusammen
  - Eine modulare Mittelschicht bindet unter Umständen nicht alle Legacy Module zusammen
    - **Best case: Gruppierung der LGMs durch Backend führt zu wenigen Abhängigkeiten**
    - **Worst case: monolithische Mittelschicht...**

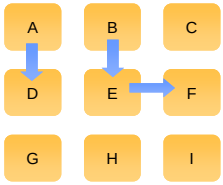
Abhängigkeit zwischen Legacy Programmen



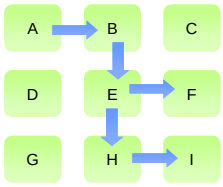
A benötigt: D
B benötigt: E
C benötigt: -
D benötigt: -
E benötigt: F
usw.

- Einzelne Legacy Programme sind abhängig voneinander
- Eine Programm Aktualisierung bedingt weitere Aktualisierungen
- Pflege aller vorhandenen Abhängigkeiten in Tabelle nötig
- Größenordnung hängt von Anzahl der Programmen ab

Abhängigkeit zwischen Legacy und Java Programmen



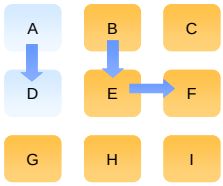
- Legacy funktioniert



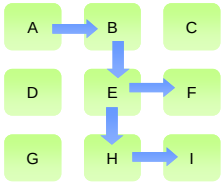
- Gleiches gilt für die Java Seite!

Minimaler Update-Set bei Abhängigkeit zwischen Legacy Programmen

Orientation in Objects



- Legacy und Java Programme sind ebenfalls voneinander abhängig

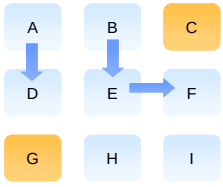


- Legacy Änderung bedingt nun Java Änderungen (eventuell zyklisch!)

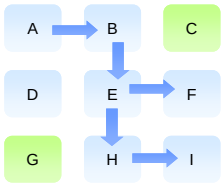
© 2010 Orientation in Objects GmbH

Minimaler Update-Set bei Abhängigkeit zwischen Legacy und Java Programmen

Orientation in Objects



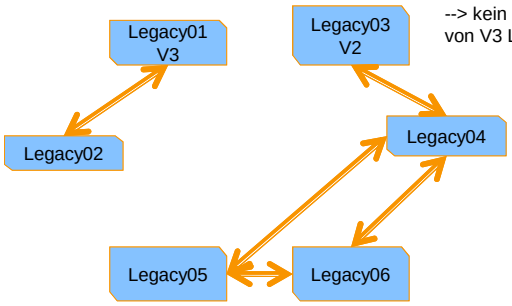
- Abhängigkeitspflege wird komplexer und fehleranfälliger
- Fehlen von Services führt zu vielen kleinen Programmen
- Kleinste Fehler bei Pflege der Abhängigkeiten potenzieren sich



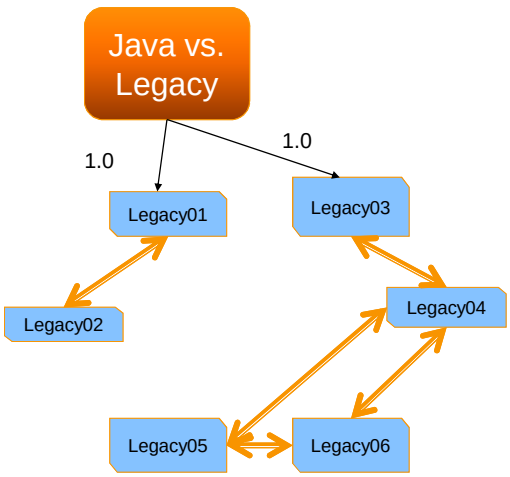
© 2010 Orientation in Objects GmbH

World without Java

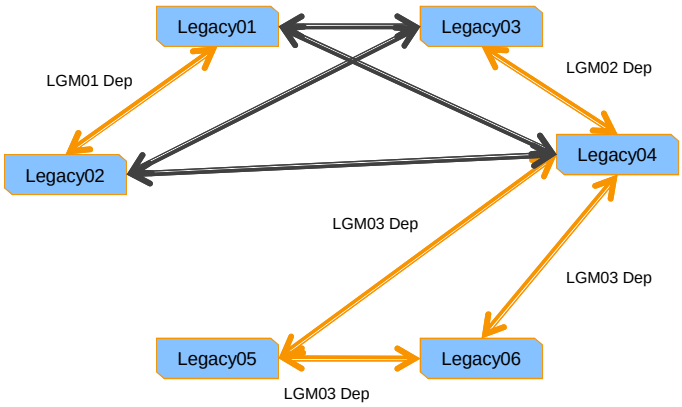
Bugfix an Legacy03
nötig
--> V2 Legacy03
--> kein Deployment
von V3 Legacy01 nötig



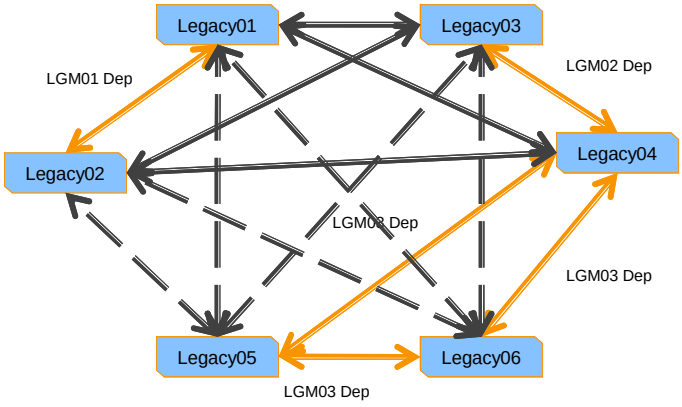
Vermittlung der Dependency durch Java vs. Legacy



Direkte Mittelschicht-Dependencies zwischen Java und LGM Programm



Was eigentlich alle zu allen bindet...

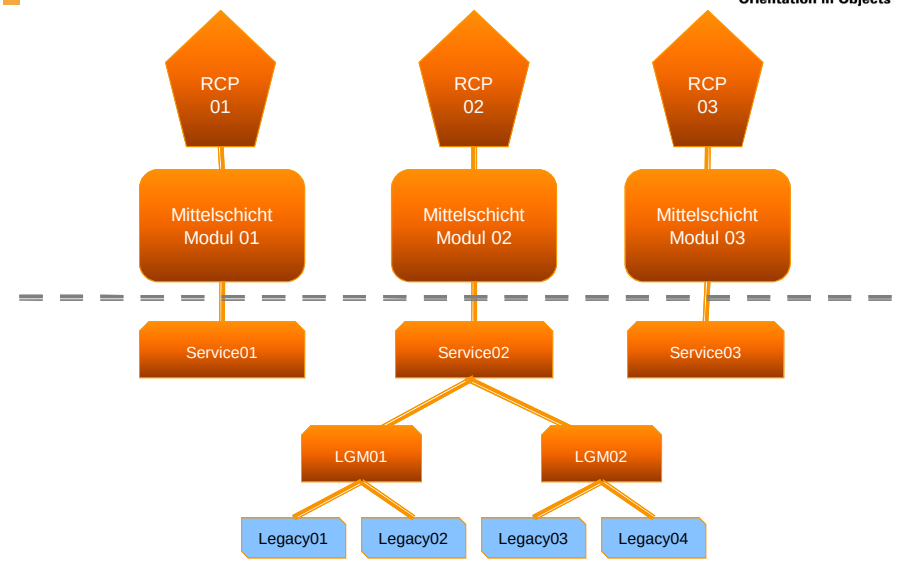


Java Dependency Management

- Legacy Bausteine werden möglichst gleich groß in Java Plugins zugeordnet und umgesetzt
- Neue Dependencies werden über Java zu Rückkopplung ins Legacy führen

- Vorteile:
  - theoretisch die Fortsetzung der bisherigen Arbeitsweise in Java
- Nachteile:
  - Java Startup und Buildzeiten sehr langsam
  - Entwicklungskosten steigen mit der Zahl der Plugins
  - Hoher QM Aufwand und hohe Fehleranfälligkeit
  - Hohes Risiko das Mittelschichts-Dependencies zu doch größeren Updates führen
  - entsprechende Versions-verwaltung existiert nicht

Konsequenz: Separation von Legacy Services

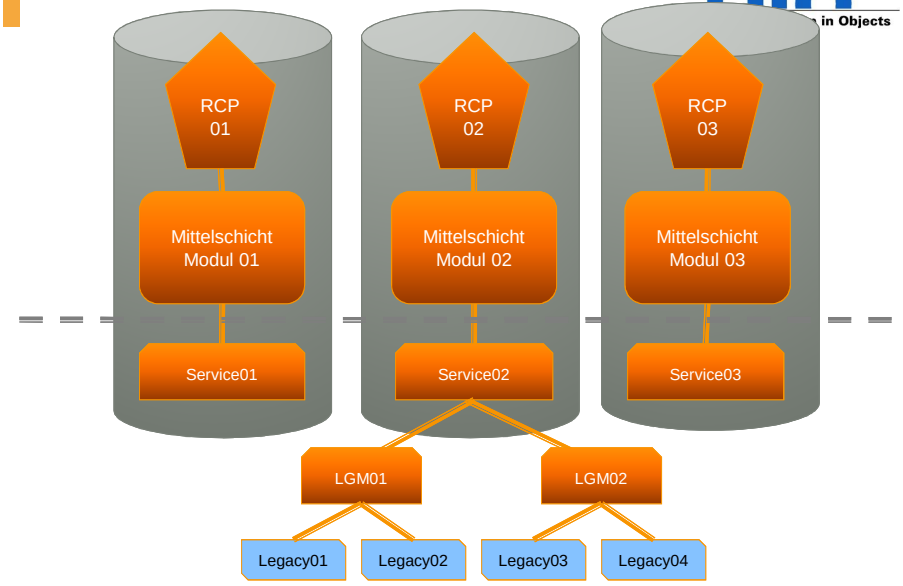


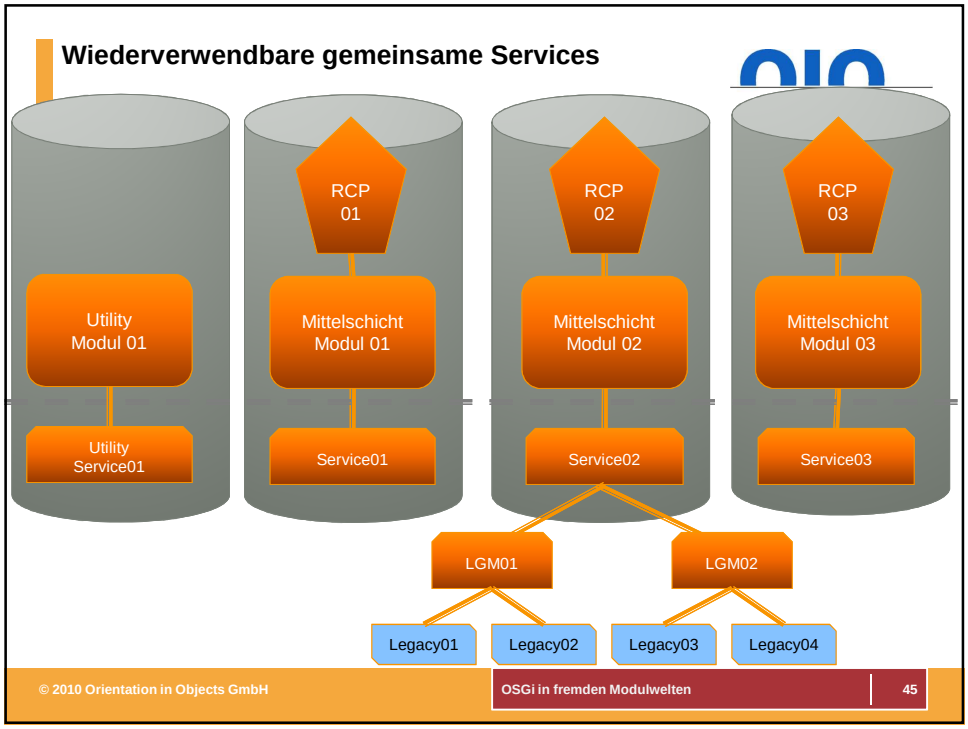
Gliederung

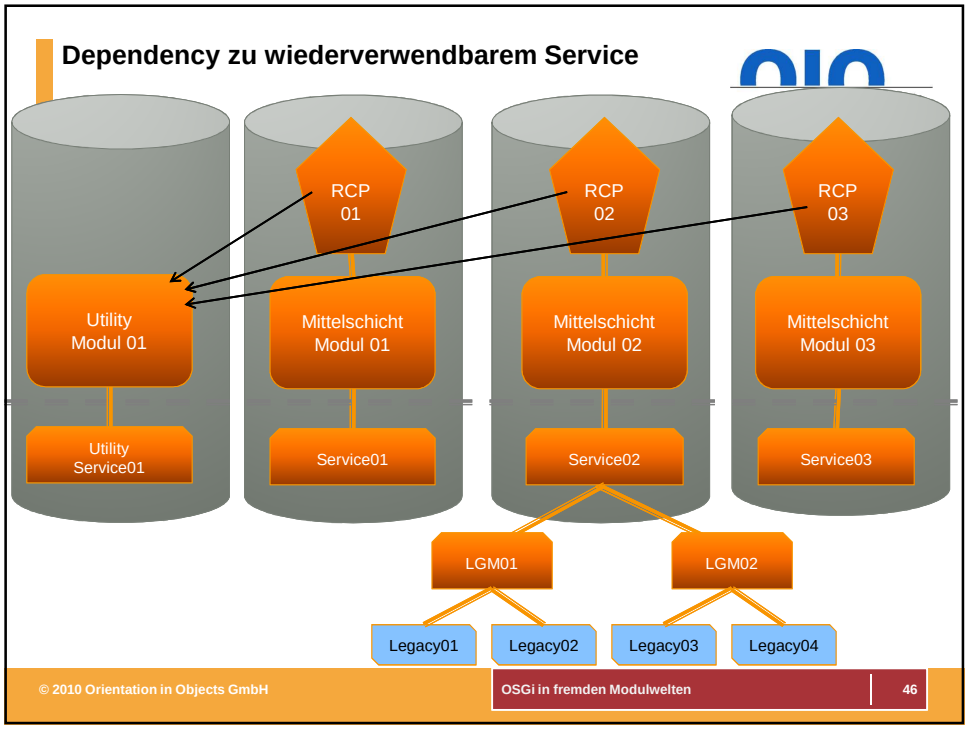
- Problemstellung
- **Lösungsansätze**
- Praktische Umsetzung

- Service Granularität
- Dependency Management
- **Silokonzept**

Silo Definition




Wiederverwendbare gemeinsame Services
RCP 01
RCP 02
RCP 03
Utility Modul 01
Mittelschicht Modul 01
Mittelschicht Modul 02
Mittelschicht Modul 03
Utility Service01
Service01
Service02
Service03
LGM01
LGM02
Legacy01
Legacy02
Legacy03
Legacy04
© 2010 Orientation in Objects GmbH
OSGi in fremden Modulwelten
45


Dependency zu wiederverwendbarem Service
RCP 01
RCP 02
RCP 03
Utility Modul 01
Mittelschicht Modul 01
Mittelschicht Modul 02
Mittelschicht Modul 03
Utility Service01
Service01
Service02
Service03
LGM01
LGM02
Legacy01
Legacy02
Legacy03
Legacy04
© 2010 Orientation in Objects GmbH
OSGi in fremden Modulwelten
46

Rückwärtskompatibilität in Services

Orientation in Objects

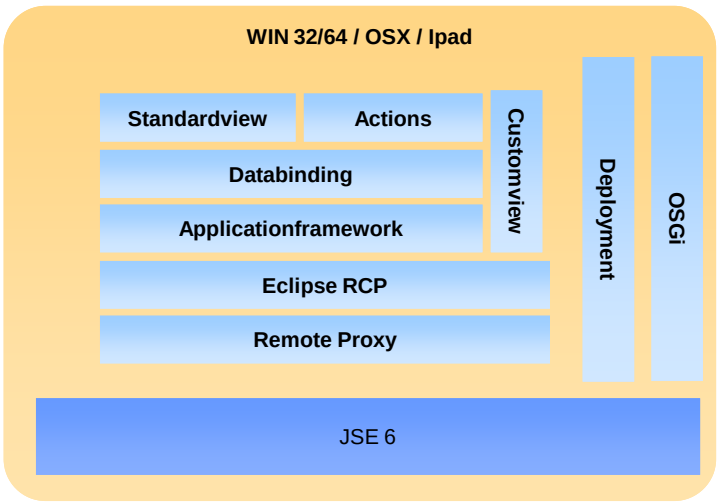
- Methoden (Funktionen) in Services sind veröffentlicht und werden weder
  - in der Signatur
  - noch in der Funktionalität geändert
- Es dürfen jederzeit neue Methoden (Funktionen) hinzugefügt werden
- Notwendige Änderungen in der Signatur oder Funktionsweise einer Methode (Funktion) werden durch hinzufügen neuer Methoden realisiert.
  - Die „alten“ Methoden werden in der Schnittstelle als „deprecated“ gekennzeichnet und werden evtl. bei dem nächsten Release entfernt.
  - Das Legacy Team muss das „deprecaten“ von Methoden (Funktionen) kommunizieren.

Gliederung

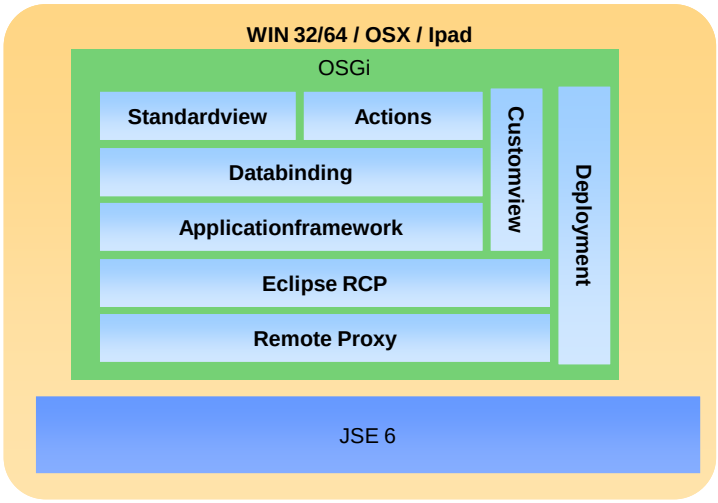
Orientation in Objects

- Problemstellung
- Lösungsansätze
- **Praktische Umsetzung**
  - **Laufzeitarchitektur**
  - Buildmanagement
  - Releasemanagement

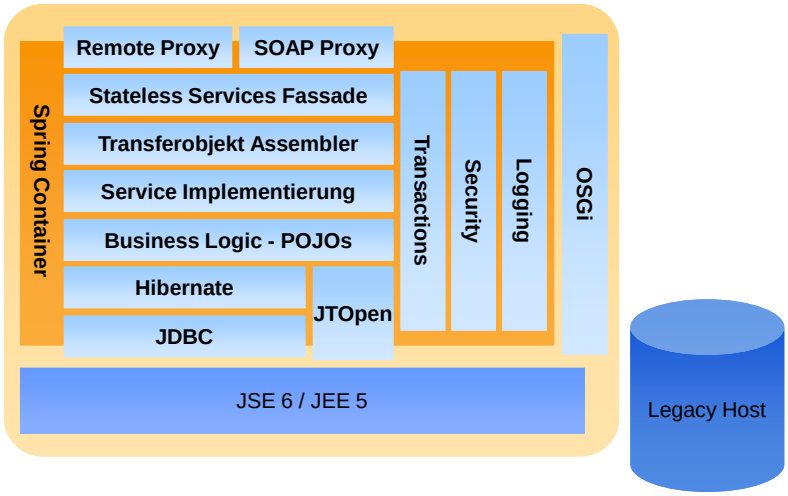
Client



Client (reworked ?)

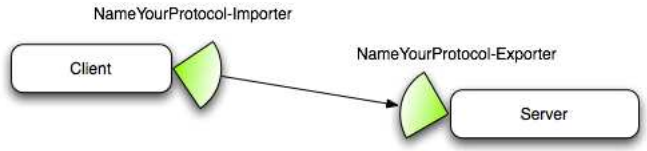


Middletier

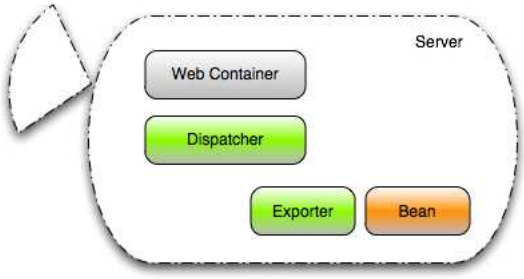


Remoting Classic View

Remoting Spring Style



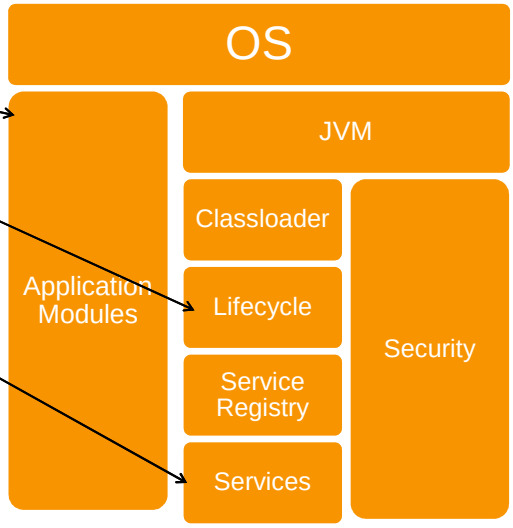
Remoting Spring Style (Detailed)



Benutzte OSGi - Layers

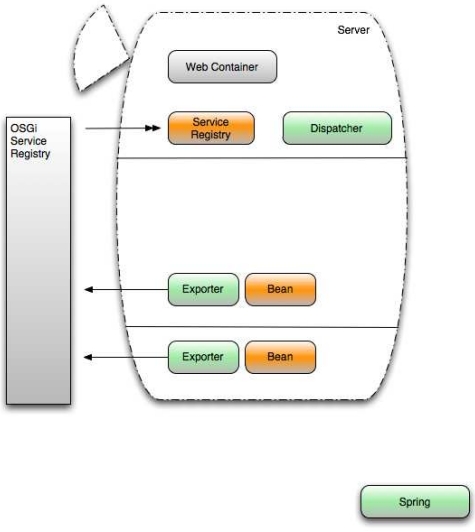
Orientation in Objects

- Component Definition
- Component Management
- Host Remoting



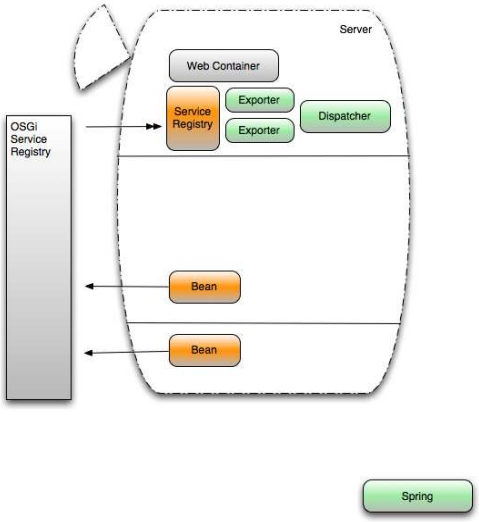
Remoting Spring Style (Detailed) OSGified

Orientation in Objects



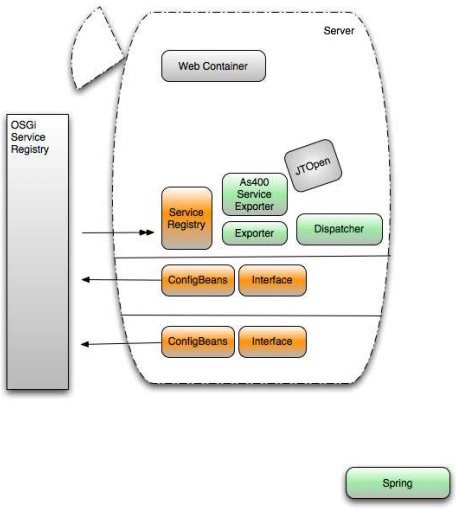
Remoting Spring Style (Detailed) OSGified

Orientation in Objects



Host Connector

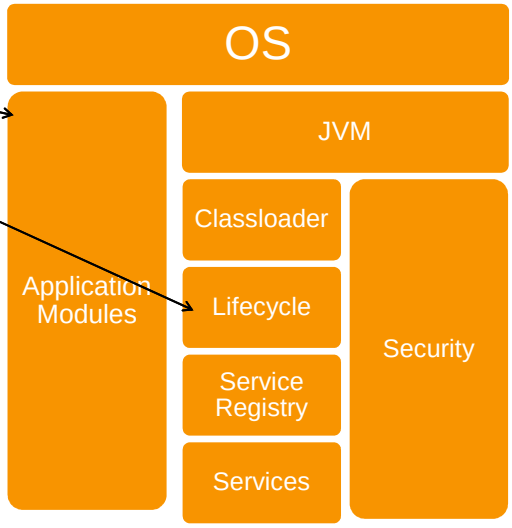
Orientation in Objects



Benutzte OSGi - Layers

Orientation in Objects

- Component Definition
- Component Management



Gliederung

Orientation in Objects

- Problemstellung
- Lösungsansätze
- **Praktische Umsetzung**

- Laufzeitarchitektur
- **Buildmanagement**
- Releasemanagement

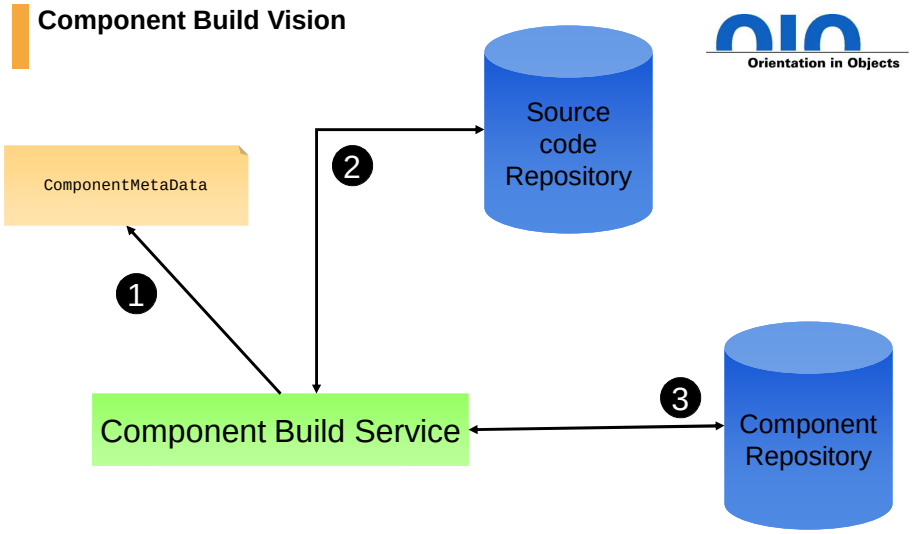
Vision Build-Management

Orientation in Objects

- Build der Komponenten-Versionen
- Zuordnung von Silo-Versionen zu Komponenten-Versionen
- Generierung einer Liste von Silo-Versionen aus dem Legacy-System
- Konstruktion des lauffähigen Products

Component Build Vision

Orientation in Objects



Orientation in Objects

Sieht doch gar nicht so kompliziert aus...

Orientation in Objects

Ansätze im Java/ OSGi-Umfeld

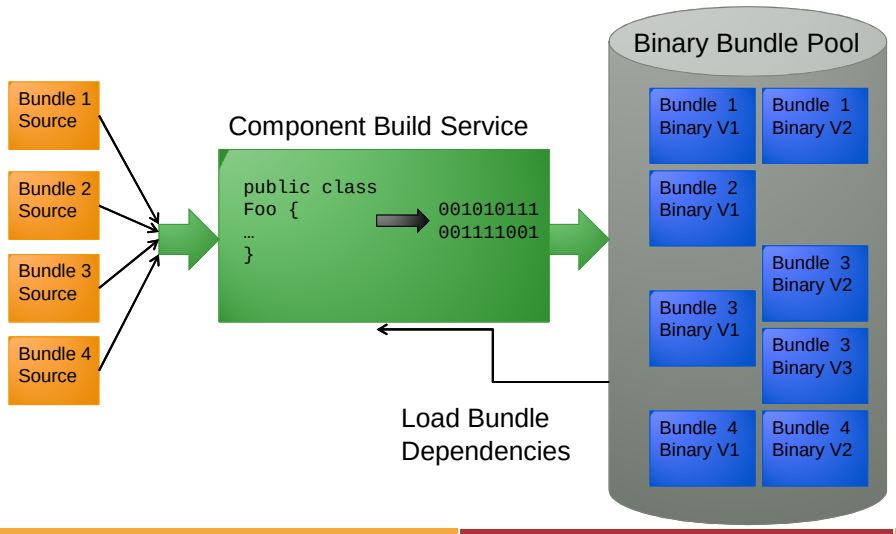
Orientation in Objects

- Buildwerkzeuge
  - PDE Build
  - Ant
    - **Sigil**
  - Bundlor
  - BND
  - Maven
    - **Tycho**
  - buckminster
- Artefakt-Management
  - OSGi Bundle Repository (OBR)
  - p2
  - Ivy
- IDE
  - Integration mit Build- und Artefakt-Management
  - Start und Test der Anwendung ohne Build/Deploy Zyklus

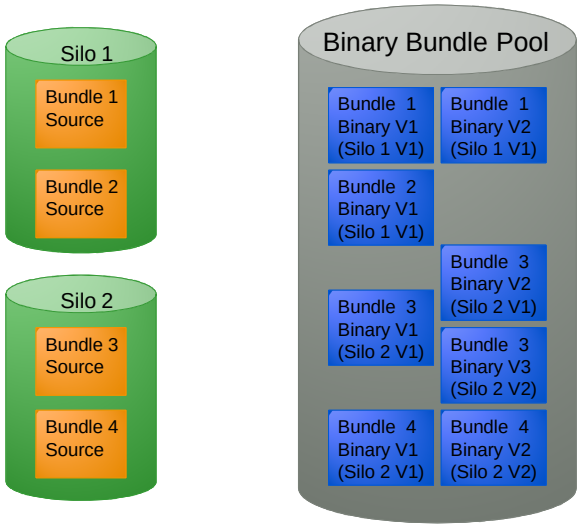
Buildinfrastruktur

Orientation in Objects

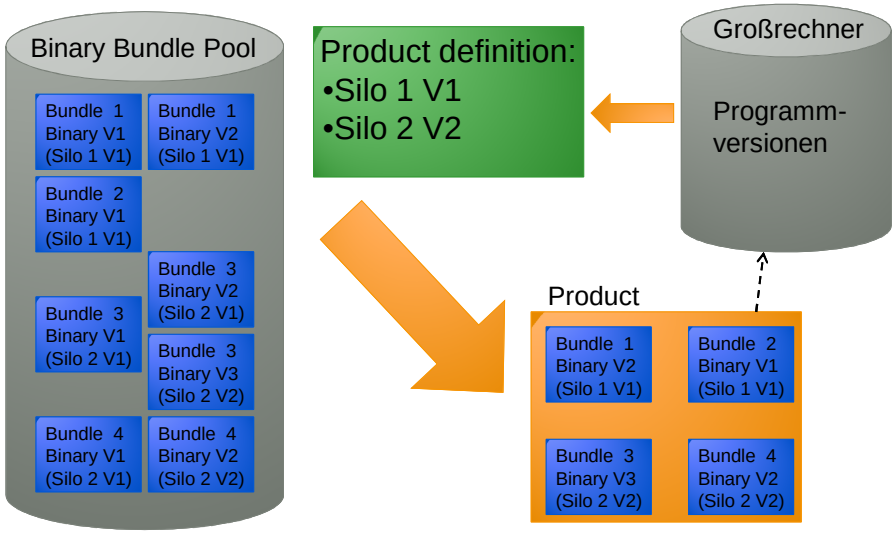
Component Build



Zuordnung der Silos und Bundles



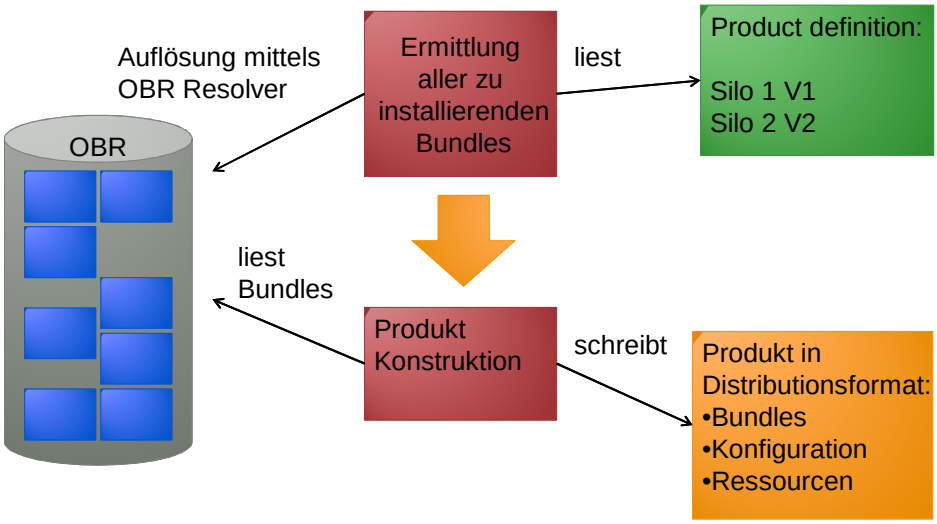
Konstruktion des kompatiblen Produktes



Lösungsidee Build-Prozess

- Lösung mit Hilfe eines OSGi Bundle Repositorys (OBR)
  - Zuordnung der Silo-Versionen über OBR-Metadaten
  - Ermittlung der neusten kompatiblen Versionen mittels OBR-Abfragen
- Einspielen der Artefakte in das OBR während des Build-Prozesses
  - Bspw. mittels Maven2
- Konstruktion der Produktes aus den ermittelten Bundles
  - PDE-Build
  - Ant/ Ivy
  - Maven2
  - Zusatzanwendungen/ Speziallösung

Die Konstruktion des Produktes aus dem OBR (2)



Die Konstruktion des Produktes aus dem OBR (1)

- Ermittlung der Bundles
- Benutzung eines OBR Resolvers
- Parametrisierung mit allen benötigten Silos
- Auflösung der Abhängigkeiten durch den Resolver
- Abfrage aller zu installierenden Bundles

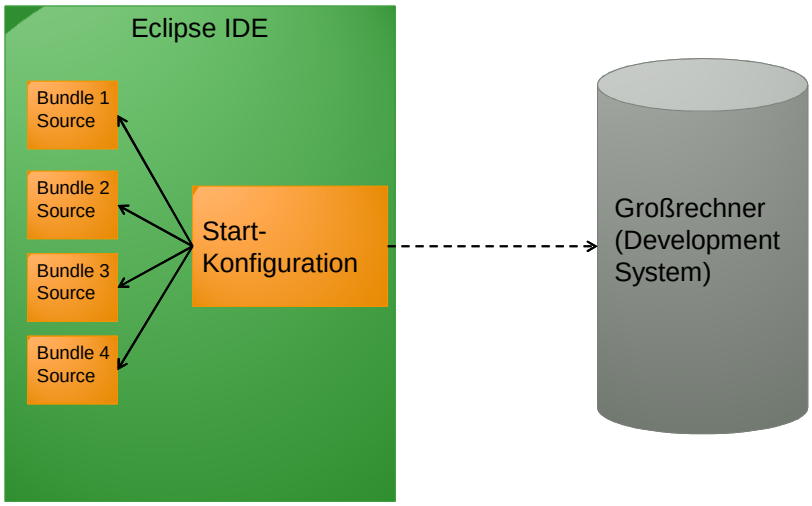
Randbedingungen durch den Entwicklungsprozess

Orientation in Objects

- Einfacher Startmechanismus aus der IDE
- Mehrere Silos zusammen startbar
- Start/Debugging von bestimmten Silo-Versionen aus der IDE

Product-Start beim Head-Development

Orientation in Objects



Starten der Anwendung aus der IDE

Orientation in Objects

- Keine Notwendigkeit eines Build/Deploy Zyklus
  - Änderungen eines Bundles werden direkt für die Abhängigen sichtbar
  - Produktivität
  - Ausführen des Head-Standes ohne Deployment
  - Hot Code Replacement beim Debugging nutzbar
  - Startup aus IDE ist anders gestaltet, als Binär-Product
- Zugriff auf Binarys
  - Test bestimmter Silo-Versionen ohne Konstruktion des Produktes
  - Anbindung der IDE an Component Repository notwendig

IDE-Integration von Binaries

Orientation in Objects

- Nativer Support in Eclipse über Target Platform
  - Binaries müssen lokal verfügbar sein
- Support eines OBR z.B. mittels Sigil Eclipse Plugin
  - Mögliches Format des Component Repositories
  - Remote ansteuerbar
- bindex
  - Erzeugung von OBR Metadaten aus Bundles
  - http://www.osgi.org/Repository/BIndex
- Apache Felix Maven Bundle Plugin
  - Deployment von Artefakten in ein OBR
  - Konfiguration durch die pom.xml
  - Liest keine Artefakte aus einem OBR
  - http://felix.apache.org/site/apache-felix-maven-bundle-plugin-bnd.html

Gliederung

- Problemstellung
- Lösungsansätze
- **Praktische Umsetzung**
  - Laufzeitarchitektur
  - Buildmanagement
  - **Releasemanagement**

Herausforderungen

Orientation in Objects

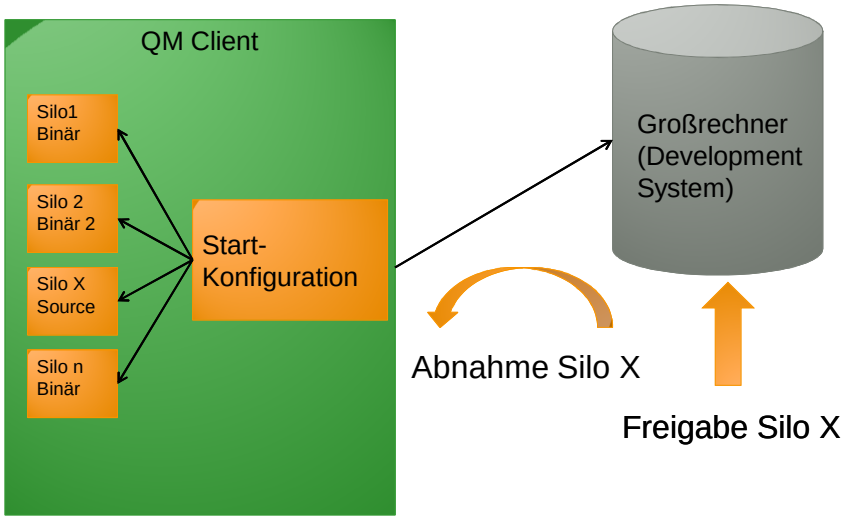
- Deployment
  - p2
  - Webstart
- Metatadaten Management
  - Pflege der MANIFEST.MF durch Entwickler
  - Generierung der MANIFEST.MF
- Verknüpfung von Java Silos und Legacy Services

Release Schema

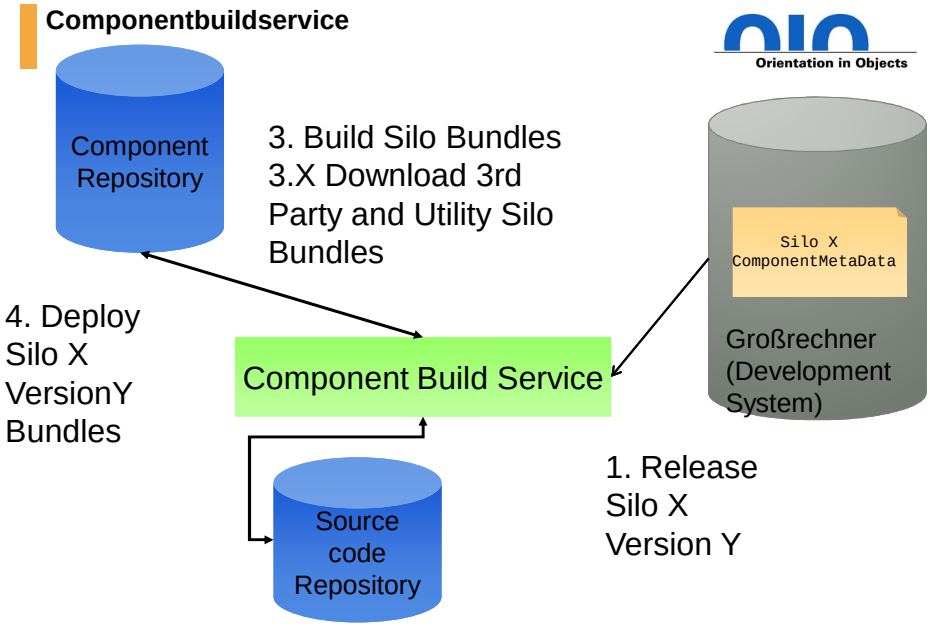
Orientation in Objects

- Major Version
  - 1 pro Jahr
  - Kunde muss vertraglich migrieren um Support zu erhalten
  - Persistenzlayer änderbar
  - Geplante 3rd Party Libraries Änderungen
- Minor Version
  - Neue Version eines LGM
- Bugfix Release
  - reiner Bugfix innerhalb Java
  -

Releasefreigabe minor Release



Componentbuildservice



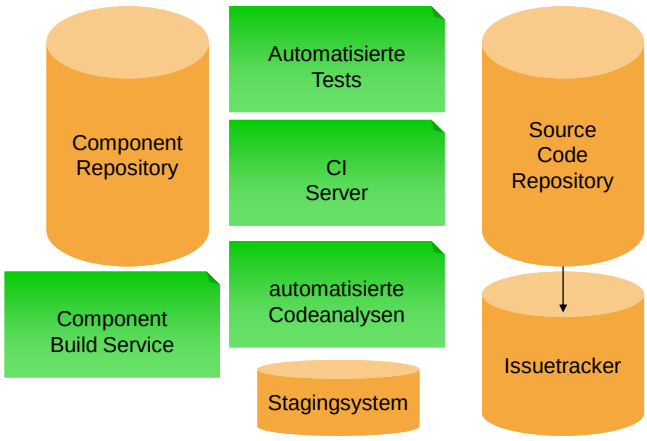
QM-Infrastruktur

Orientation in Objects

- Hoher Integrationsgrad birgt Qualitätsrisiko
  - daher automatisierte Integrationstest als Regression
- Verbesserung der Codequalität durch gezielte Massnahmen nach Messungen
  - Integration von Codemetriken in den Buildprozess
- Zusammenfassung in Continuous Integration Server
- bei größeren Customizings birgt Datenbasis des Kunden ebenfalls Qualitätsrisiko
  - Einsatz eines Staging Systems
- Dokumentation der Auswirkungen von Entwicklungsaktivitäten
  - Kopplung von Issuetracker und Sourcecode Repository
  - Analyse und automatisierte Überwachung

QM Infrastruktur

Orientation in Objects



Orientation in Objects

Fragen ?

Orientation in Objects GmbH

Weinheimer Str. 68
68309 Mannheim

www.oio.de
info@oio.de

Mehr von OIO zum Thema...

Orientation in Objects

- Schulung: Einführung in die OSGi Service Platform
  - http://www.oio.de/seminar/java/seminar-osgi-schulung-equinox-traing.htm
- Beratung: Open Source Plattformen wie OSGi, Spring, RCP…
  - http://www.oio.de/beratung-consulting/open-source-software/plattform/index.htm
- Veröffentlichung: Flexible Plattformen - Open Source Portale im Vergleich
  - http://www.oio.de/public/opensource/t3n_nr8_portalvergleich.pdf
- Referenz: Aufwandsschätzung: Migrationsprojekt bei Finanzdienstleister
  - http://www.oio.de/referenzen/competence-center/referenz-beratung-aufwandsschaetzung-migration-grossprojekt.htm

© 2010 Orientation in Objects GmbH

Orientation in Objects

Vielen Dank für ihre Aufmerksamkeit !

Orientation in Objects GmbH

Weinheimer Str. 68
68309 Mannheim

www.oio.de
info@oio.de