

Anwendungsintegration
mit REST-Services

ex|Xcellent
solutions

OOP 2010 - 27.01.2010
Dr. Ralph Guderlei



Agenda

x| Das Problem

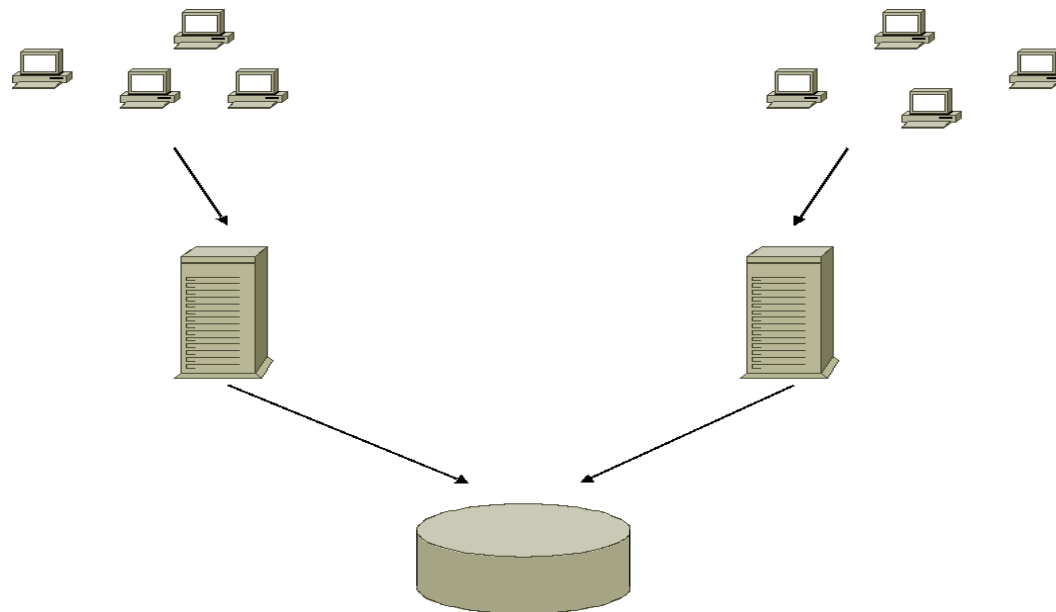
x| REST

x| JAX-RS und Jersey

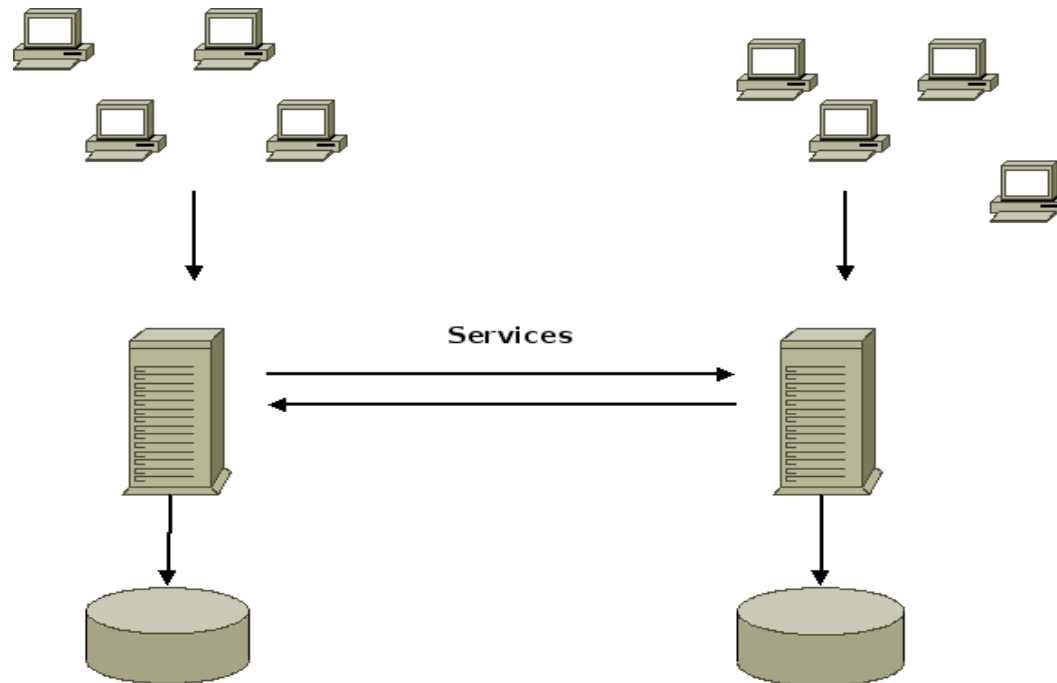
x| Die Lösung

x| Fazit

- x| zwei autonome Systeme „die Inventar-Datenbank“ und „die Workflow-Plattform“
- x| Teilweise gemeinsame Datenhaltung (z.B. Benutzerverwaltung) (Shared Database Pattern)



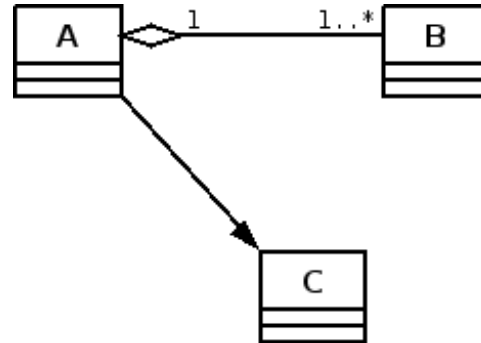
- x| Getrennte Datenhaltung
- x| Integration über Services (→ Remote Procedure Invocation Pattern)



- x| Representational State Transfer
- x| Menge an Bedingungen für eine Architektur eines verteilten Systems
 - | Client-Server
 - | Stateless
 - Aufrufe müssen alle Informationen beinhalten, die zum Verarbeiten des Aufrufs notwendig sind
 - | Cache
 - | Uniform Interface
 - URIs
 - Manipulation von Ressourcen durch Repräsentationen
 - sich selbst beschreibende (standardisierte) Methoden
 - HATEOAS
 - | Layered System
 - | Code on Demand

REST ist NICHT

- x| XML over HTTP
- x| Alle Parameter eines Aufrufs werden in den URI codiert
- x| Simples CRUD auf Daten



- x| Relationen in Objektmodellen können abgebildet werden durch
 - | Den URI einer Resource
`http://example.com/a/{identifizier}/b/`
 - | Links/URIs in Repräsentationen
- x| → gewisse Ähnlichkeiten erkennbar

x| Wiederverwendung von HTTP-Features

- | Authentifizierung (HTTP Auth)
- | Caching
- | Verschlüsselte Kommunikation (HTTPS)

x| Beschreibung von REST-Service mit WADL

- x| Standardisiert in JSR-311 (final seit Oktober 2008)
- x| Implementierungen: Jersey, RESTEasy, Apache CXF, Restlet
- x| Programmiermodell: POJO + Annotations
- x| Deklaration von Services über standardisierte Annotations
- x| Automatisches Marshalling/Unmarshalling entsprechend dem definierten MIME-Type

@GET, @POST, @PUT, @DELETE, ...	Definieren die HTTP-Methode
@Produces, @Consumes	Definieren den MIME-Type des Requests bzw. der Response
@Path	Definiert die URI der Ressource
@PathParam, @HeaderParam	Erlauben das Auslesen von URI- Parametern bzw. Parametern aus dem HTTP-Header

```
3 import javax.ws.rs.GET;
4 import javax.ws.rs.Path;
5 import javax.ws.rs.PathParam;
6 import javax.ws.rs.Produces;
7 import javax.ws.rs.WebApplicationException;
8 import javax.ws.rs.core.MediaType;
9 import javax.ws.rs.core.Response;
10
11 @Path("/myservice")
12 public class MyServiceRessource {
13     @GET
14     @Path("/{identifier}")
15     @Produces({MediaType.APPLICATION_XML, MediaType.APPLICATION_JSON})
16     public MyClass getObject(@PathParam("identifier") String identifier){
17         MyClass myObject = MyDao.findById(identifier);
18         if(myObject == null){
19             throw new WebApplicationException(Response.Status.NOT_FOUND);
20         } else {
21             return myObject;
22         }
23     }
24 }
25 }
```

- x| HTTP Status Codes ermöglichen einen standardisierten Austausch von Informationen über Erfolg/Misserfolg eines Aufrufs
 - | Zusätzlich können Applikationsspezifische Fehlertexte übertragen werden
- x| Frameworks verwenden Server- und Clientseitig „normale“ Java-Exceptions

- x| Minimale Anforderungen an Infrastruktur (nur Servlet-Container)
- x| Standardisierte API, mehrere Implementierungen
- x| Einfache Technologie
- x| Wenig Abhängigkeiten
- x| Keine Probleme mit Firewalls, da HTTP/HTTPS
- x| Strukturierung von URIs analog zum Objektmodell möglich

- x| Entscheidung für Jersey als JAX-RS Implementierung
- x| Integration in bestehende Anwendungen unter Java 1.5 + WebSphere, Tomcat 6 und WebStart-Clients
- x| Java - only - Ansatz: POJOs für Datenaustausch, keine XSD für Repräsentationen o.ä.

x| Vorgehensweise:

- | Identifikation der benötigten Informationen / Operationen
- | Entwurf der Ressourcen für die benötigten Informationen:
Strukturierung analog zum bestehenden Objektmodell
→ API
- | Entwurf der Repräsentationen:
Auswahl der benötigten Attribute/Relationen, Erstellung von DTOs
Wahl der Darstellung (XML, JSON, PDF, ...?)

x| „eager loading“ vs. „lazy loading“

- | „eager loading“: referenzierte Objekte werden mit serialisiert
- | „lazy loading“: URL auf die Resource, die die referenzierten Objekte bereitstellt

x| „batch reading“ vs. „one by one“

- | „batch reading“: Resource liefert alle (referenzierten) Objekte
- | „one by one“: pro (referenziertem) Objekt ein Service-Call

x| Strukturierung

http://example.com/a/{identifier-a}/b/{identifier-b}

```
@Path("/a/{identifier-a}")
public class RessourceForA {
    @Path("/b")
    public RessourceForB delegate(){
        return new RessourceForB();
    }

    @GET
    public A getA(@PathParam("identifier-a") String id){
        return new A();
    }

    // ...
}

public class RessourceForB {
    @GET
    @Path("/{identifier-b}")
    public B getB(@PathParam("identifier-b") String id){
        return new B();
    }

    // ...
}
```

- x| Problem: kein Zugriff auf Servlet-Container möglich (insbesondere keine Konfiguration der Zugriffskontrolle)

- x| Lösungsansatz:
 - | Analog zu HTTP Auth werden (eigene) Authentifizierungsinformationen in die Request-Header eingefügt

 - | Auf Serverseite: Wiederverwendung der bestehenden Authentifizierungs-/Authorisierungs-Infrastruktur

 - | Nachteil: API der Service-Methoden wird unnötig aufgebläht

x| WebDAV

- | Zugriff auf System über Dateisystem
- | Erweiterung bestehender JAX-RS Services möglich (WebDAV support for JAX-RS)

x| Datei-Export über Wahl des MIME-Types (z.B. PDF, Excel)

- x| Keine Serialisierung von „Original-Objekten“: oft wird nur ein geringer Teil der Attribute benötigt
- x| Größte Fehlerquelle: client-seitige Konstruktion von Service-URLs
→ HATEOAS wichtig!
- x| Sauberer Transport von Fehlerinformationen

- x| Einfache Integration in bestehende (Web-) Anwendungen
- x| Einfache, reibungslose Verwendung von JAX-RS/Jersey
- x| Flexibel: z.B. einfache Verwendung eines eigenen Authentifizierungsmechanismus
- x| Ein bestehendes Objektmodell lässt sich einfach als Ressourcen abbilden

x| Richardson, L.; Ruby, S.(2007), RESTful Web Services, O'Reilly

x| JSR-311: <https://jsr311.dev.java.net/>

x| Sun Jersey: <https://jersey.dev.java.net/>

x| WebDAV support for JAX-RS: <https://webdav.dev.java.net/>

Vielen Dank für Ihre Aufmerksamkeit!

Fragen??

Gerne auch am Stand von
eXXcellent solutions
im Ausstellungsbereich
(Stand 8.3)

Folien bald unter <http://www.exxcellent.de>