JAVAFORUM 2007 stuttgart am 5. Juli 2007

univativ
YOUNG POTENTIALS' AGENCY

FRESH UP YOUR BUSINESS
FRESH UP YOUR BUSINESS

Integration von Web Services in J EE Anwendungen mit XFire

univativ : = Umsetzung durch
Studenten und Young Professionals.
IT- Business- Engineering- Services

Level of Services

- Support
- Outtasking / Managed Services / Outsourcing
- Consulting / Project Solutions

Facts & Figures

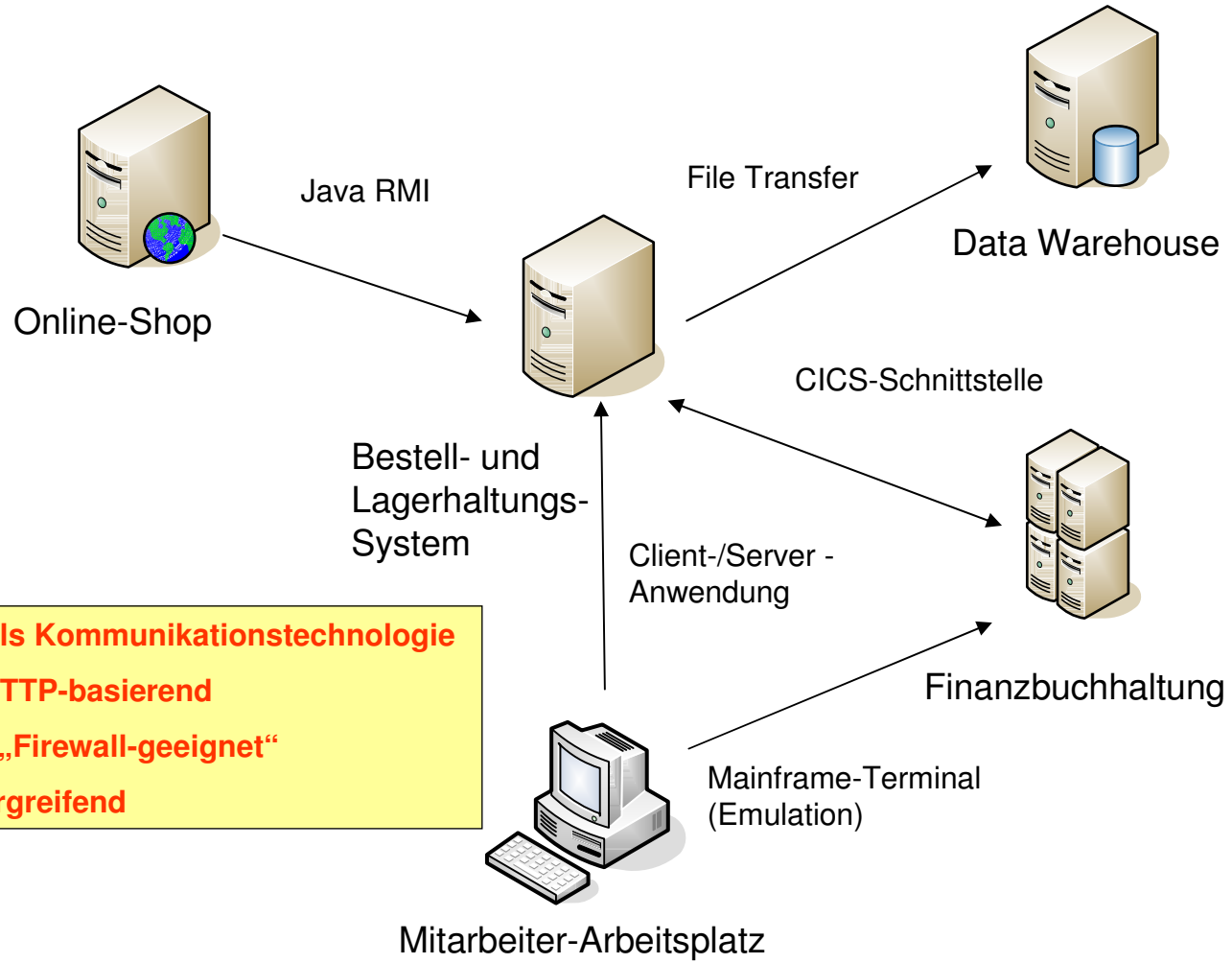
- Gründung 1996 in Darmstadt
- Mitarbeiter: ca. 320
- insgesamt ca. 700 Aufträge mit ca. 70.000 Personaltagen

univativ GmbH & Co. KG, Darmstadt, Karlsruhe, Stuttgart, Düsseldorf, München

Überblick

- Big Picture
  - EAI
  - SOA
- Web Services
  - Allgemeines
  - XFire
- Beispielszenarien
  - J EE Anwendungen verbinden
  - Web Service Providing
  - Web Service Consuming
- weitere XFire Features

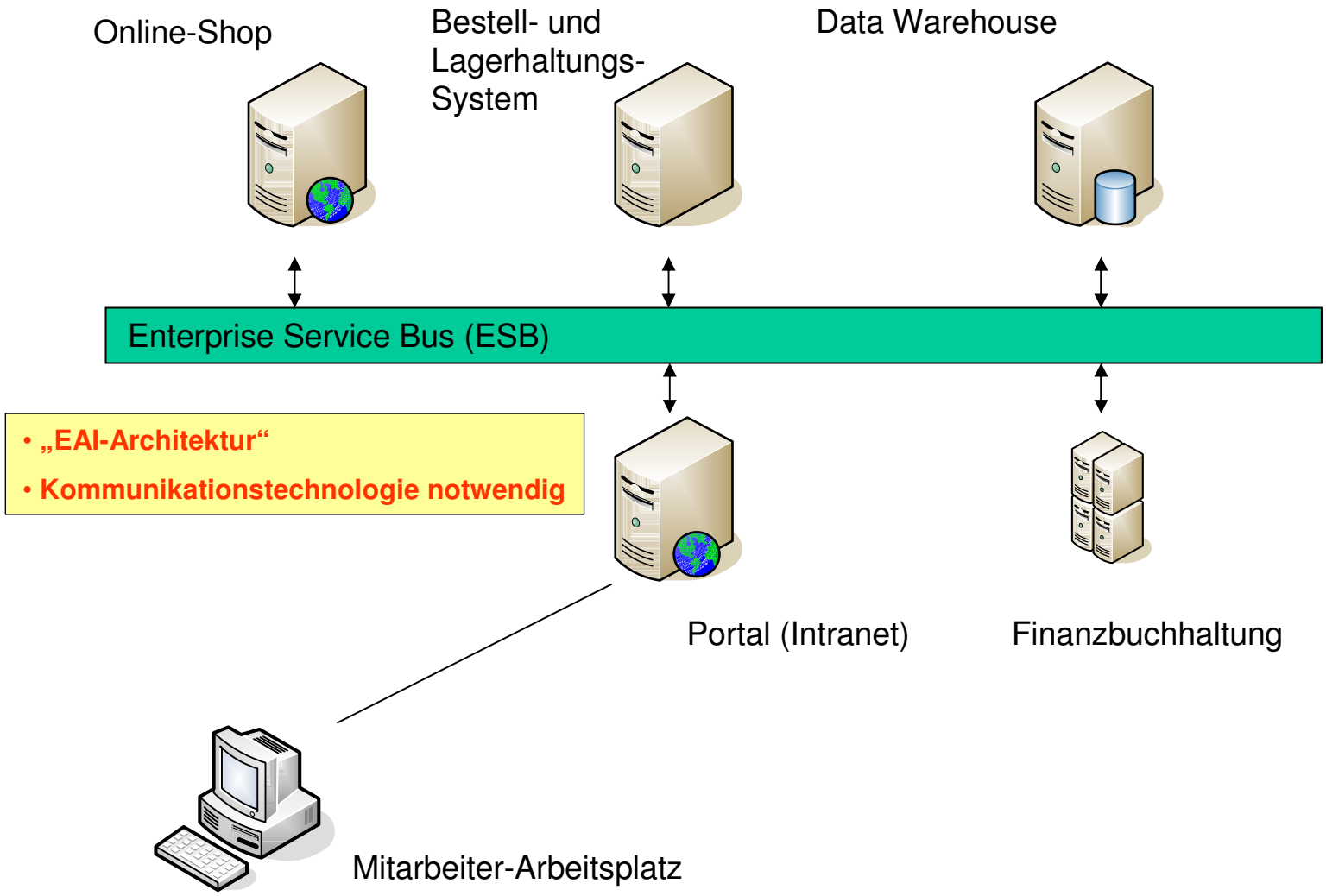
Enterprise Application Integration (EAI)



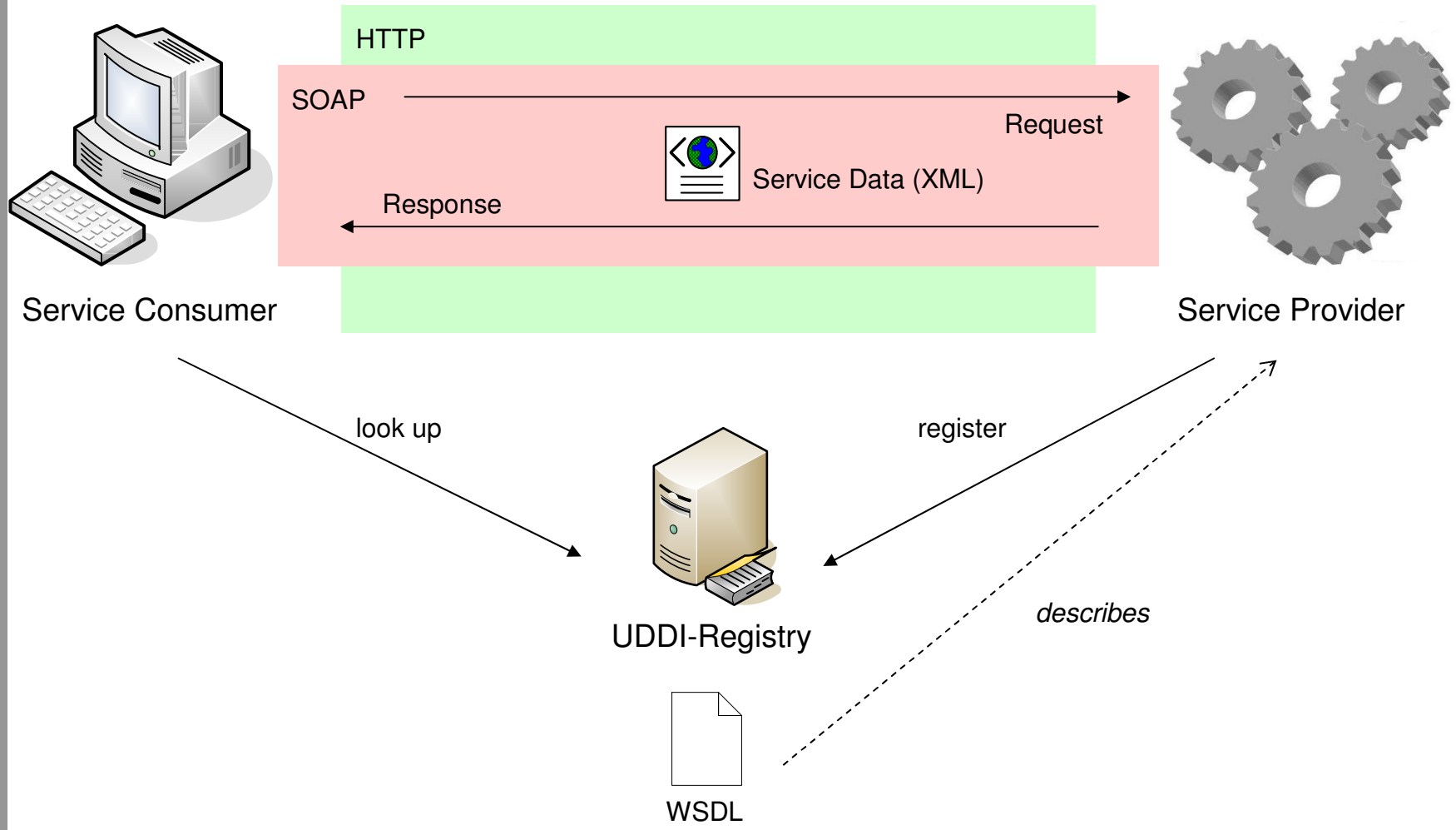
Web Services als Kommunikationstechnologie

- **TCP/IP bzw. HTTP-basierend**
- **HTTP-Port 80 „Firewall-geeignet“**
- **Plattform-übergreifend**

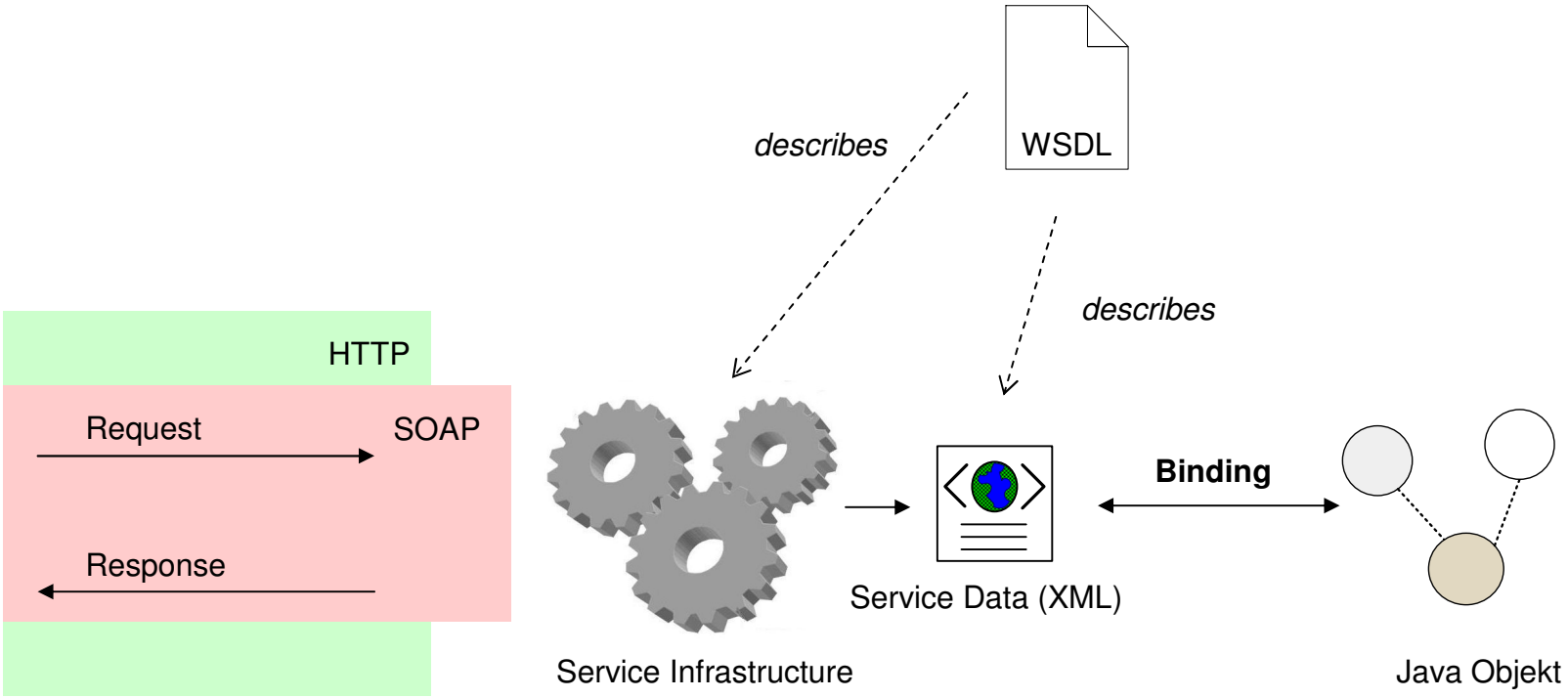
Service Oriented Architecture (SOA)



Web Services allgemein



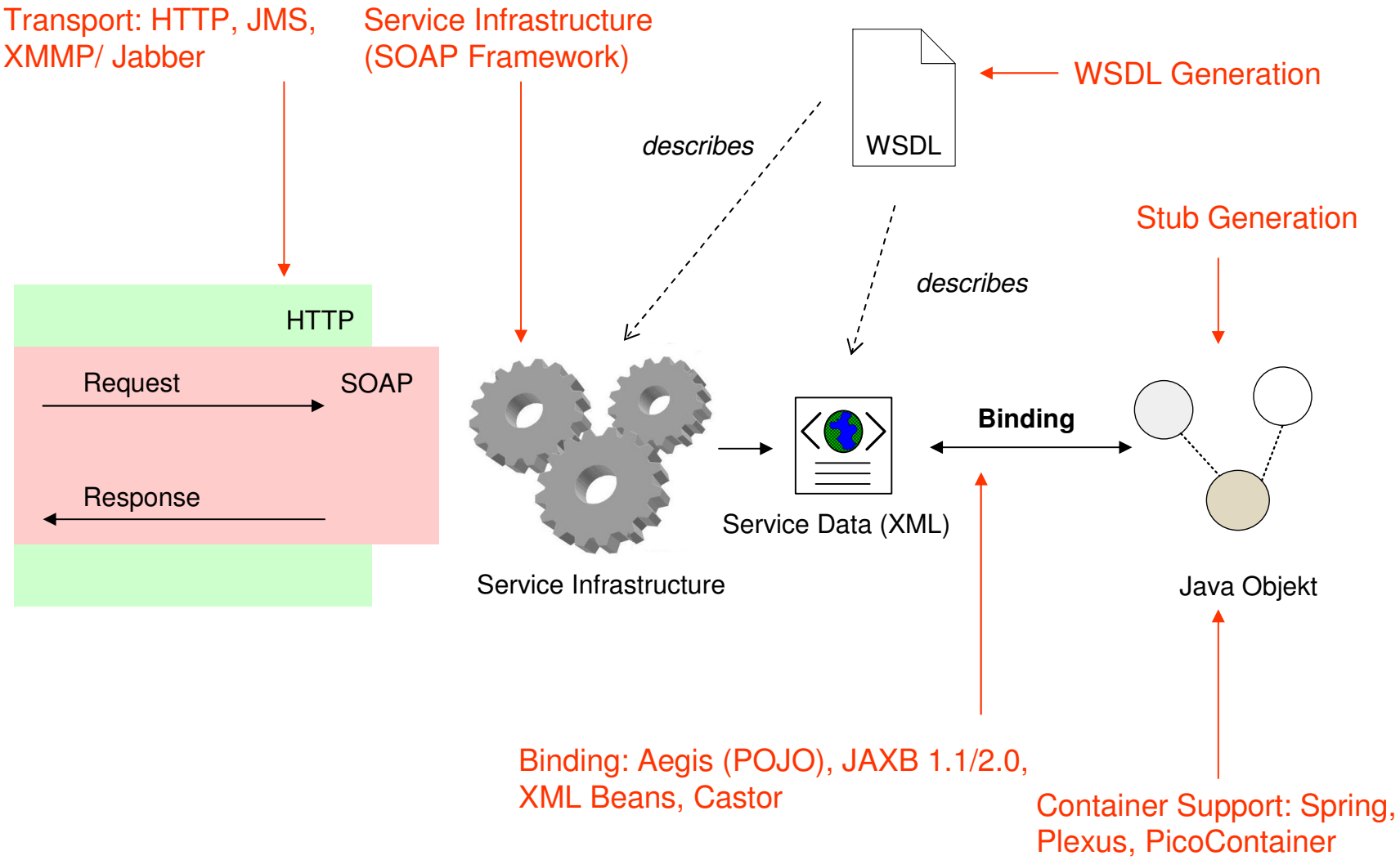
Java Web Services – Service Providing



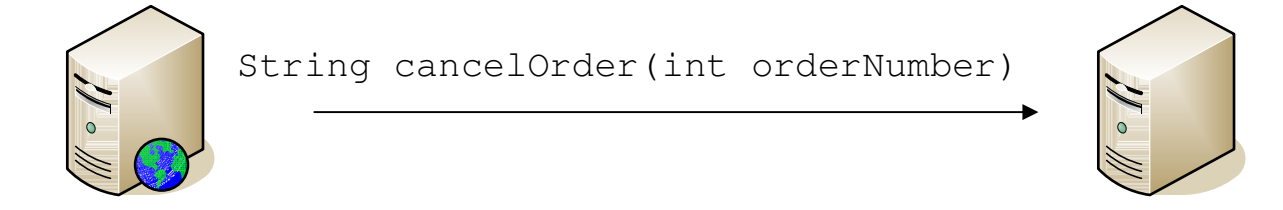
XFire Web Service Framework

- leichtgewichtiges SOAP Framework
- Open Source
- Codehaus.org Community
- Ziele
  - Einfache Verwendung
  - Performance
  - Verwendung von Standards
  - Modularität: Unterstützung austauschbarer
    - Bindings
    - Transports
    - Container

XFire - Service Providing



Szenario: J EE Anwendungen verbinden



OrderManagement.java

```
package com.mycompany.order;

public class OrderManagement
{
  public String cancelOrder(int orderNumber)
  {
    // do cancelation
    return cancelationMessage;
  }
  […]
}
```

Szenario: J EE Anwendungen verbinden

Service Provider: Refactoring…

– IOrderManagement.java

```
package com.mycompany.order;

public interface IOrderManagement
{
  public String cancelOrder(int orderNumber)

[…]
}
```

XFire POJO = JavaBean
- no args Konstruktor
- Getter/Setter
- public Methoden

– OrderManagementImpl.java

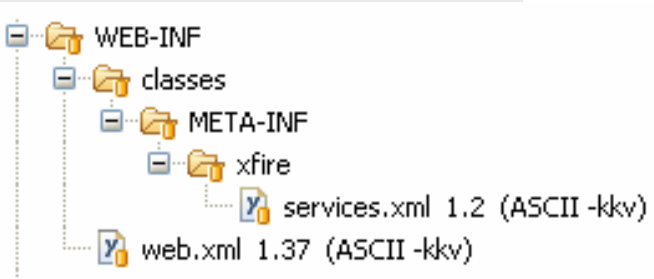
```
package com.mycompany.order;

public class OrderManagementImpl implements IOrderManagement
{
public String cancelOrder(int orderNumber)
   {
     // do cancelation
     return cancelationMessage;
   }
   […]
}
```

Szenario: J EE Anwendungen verbinden

Service Provider:

XFire-Servlet zu /WEB-INF/web.xml hinzufügen

```
<servlet>
  <servlet-name>XFireServlet</servlet-name>
  <display-name>XFire Servlet</display-name>
  <servlet-class>
    org.codehaus.xfire.transport.http.XFireConfigurableServlet
  </servlet-class>
</servlet>

[…]

<servlet-mapping>
  <servlet-name>XFireServlet</servlet-name>
  <url-pattern>/servlet/XFireServlet/*</url-pattern>
</servlet-mapping>

<servlet-mapping>
  <servlet-name>XFireServlet</servlet-name>
  <url-pattern>/services/*</url-pattern>
</servlet-mapping>
```



Szenario: J EE Anwendungen verbinden

Service Provider:

- XFire JARs zu /WEB-INF/lib hinzufügen
- /WEB-INF/classes/META-INF/xfire/services.xml erstellen

```
<beans xmlns="http://xfire.codehaus.org/config/1.0">
  <service>
    <name>OrderManagement</name>
    <serviceClass>
      com.mycompany.order.IOrderManagement
    </serviceClass>
    <implementationClass>
      com.mycompany.order.OrderManagementImpl
    </implementationClass>
  </service>
</beans>
```

Szenario: J EE Anwendungen verbinden

Service Consumer:

- XFire JARs zu /WEB-INF/lib hinzufügen
- IOrderManagement.java hinzufügen
- Proxy erzeugen und Methoden aufrufen

```
String serviceUrl =
"http://server:8080/context/services/OrderManagement";

Service serviceModel = new
ObjectServiceFactory().create(IOrderManagement.class);

XFire xfire = XFireFactory.newInstance().getXFire();
XFireProxyFactory factory = new XFireProxyFactory(xfire);

orderManagementProxy = (IOrderManagement) factory.create(serviceModel,
serviceUrl);

[…]

orderManagementProxy.cancelOrder(0815);
```

Szenario: Web Service Providing

String cancelOrder(int orderNumber)

Service Consumer

Service Provider

OrderManagement.java

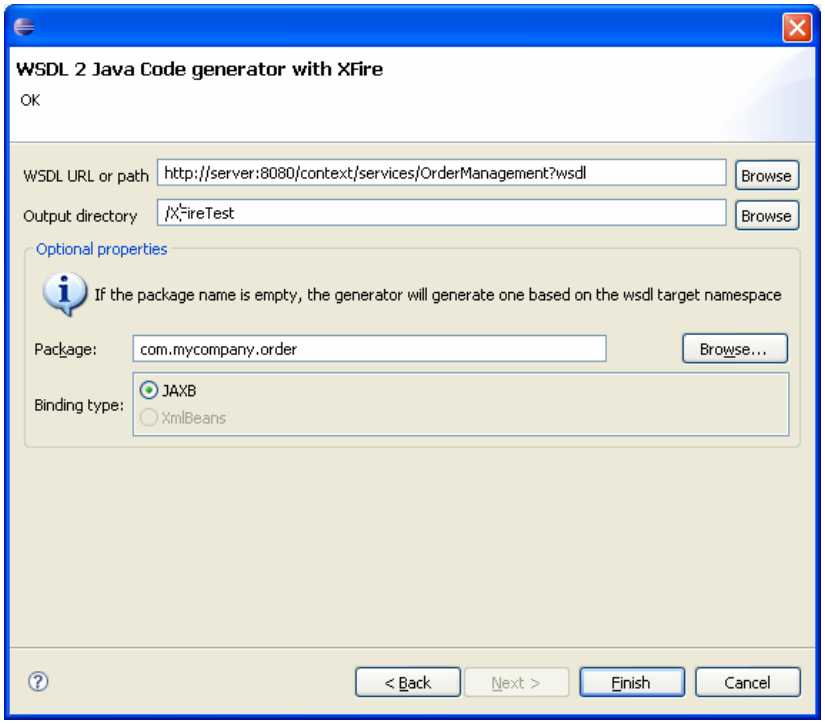
```
package com.mycompany.order;

public class OrderManagement
{
  public String cancelOrder(int orderNumber)
  {
    // do cancelation
    return cancelationMessage;
  }
  […]
}
```

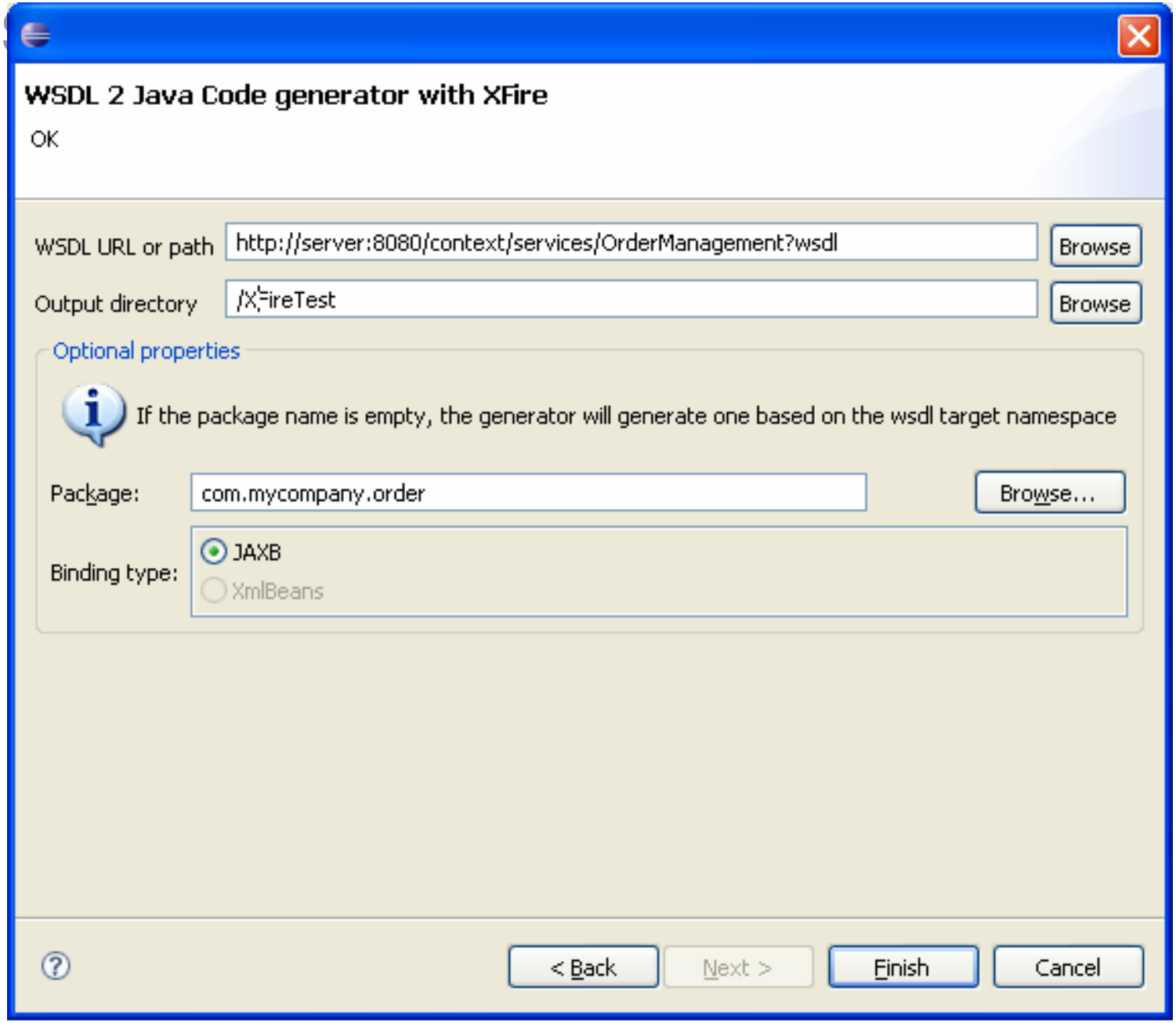
Szenario: Web Service Providing

- Service Provider:
  - XFire-Servlet zu /WEB-INF/web.xml hinzufügen
  - XFire JARs hinzufügen
  - /WEB-INF/classes/META-INF/xfire/services.xml erstellen
  - Refactoring optional:
    Aufteilung in Interface und Implementierung
- WSDL Dokument durch Aufruf erzeugen
  http://server:8080/context/services/OrderManagement?wsdl
- Client auf beliebigem Zielsystem generieren / programmieren

Szenario: Web Service Consuming

- Service Consumer:
  - XFire JARs zu /WEB-INF/lib hinzufügen
  - Java Klassen aus WSDL generieren (Eclipse Plug-in)




WSDL 2 Java Code generator with XFire
OK
WSDL URL or path
http://server:8080/context/services/OrderManagement?wsdl
Browse
Output directory
/XFireTest
Browse
Optional properties
If the package name is empty, the generator will generate one based on the wsdl target namespace
Package:
com.mycompany.order
Browse...
Binding type:
JAXB
XmlBeans
< Back
Next >
Finish
Cancel

Szenario: Web Service Consuming

- Proxy erzeugen und Methoden aufrufen

```
[…]

// generated class OrderManagement
OrderManagement orderManagementProxy = new OrderManagement();

[…]

orderManagementProxy.cancelOrder(0815);
```

- generierte Klassen kapseln:
  - Service-Endpunkt: http://server:8080/context/services/OrderManagement
  - Service-Model: (JAXB) Binding
  - XFire ProxyFactory

XFire – weitere Features

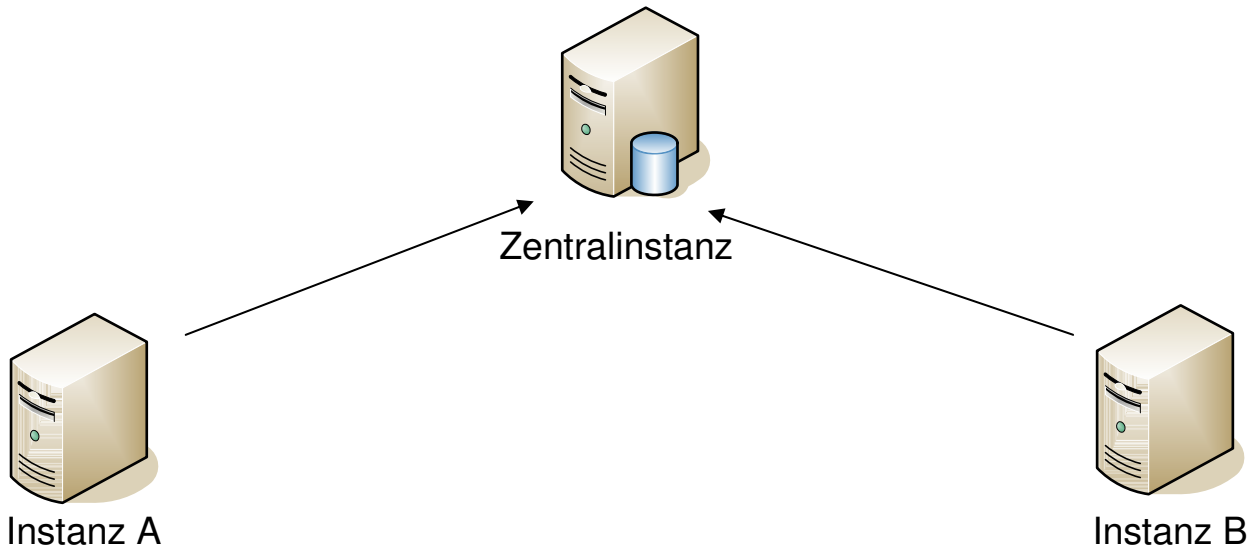
- JSR 181 Annotationen

```
package com.mycompany.order;

@WebService(name=„OrderService“)
public class OrderManagement
{

  @WebMethod(operationName="cancelOrder")
  @WebResult(name=„orderResult")
  public String cancelOrder (@WebParam(name=„orderParam“) int orderNumber)
  {
    […]
```

- Custom Handlers
- HTTP-Proxy- und SSL-Unterstützung
- build-in HTTP-Server

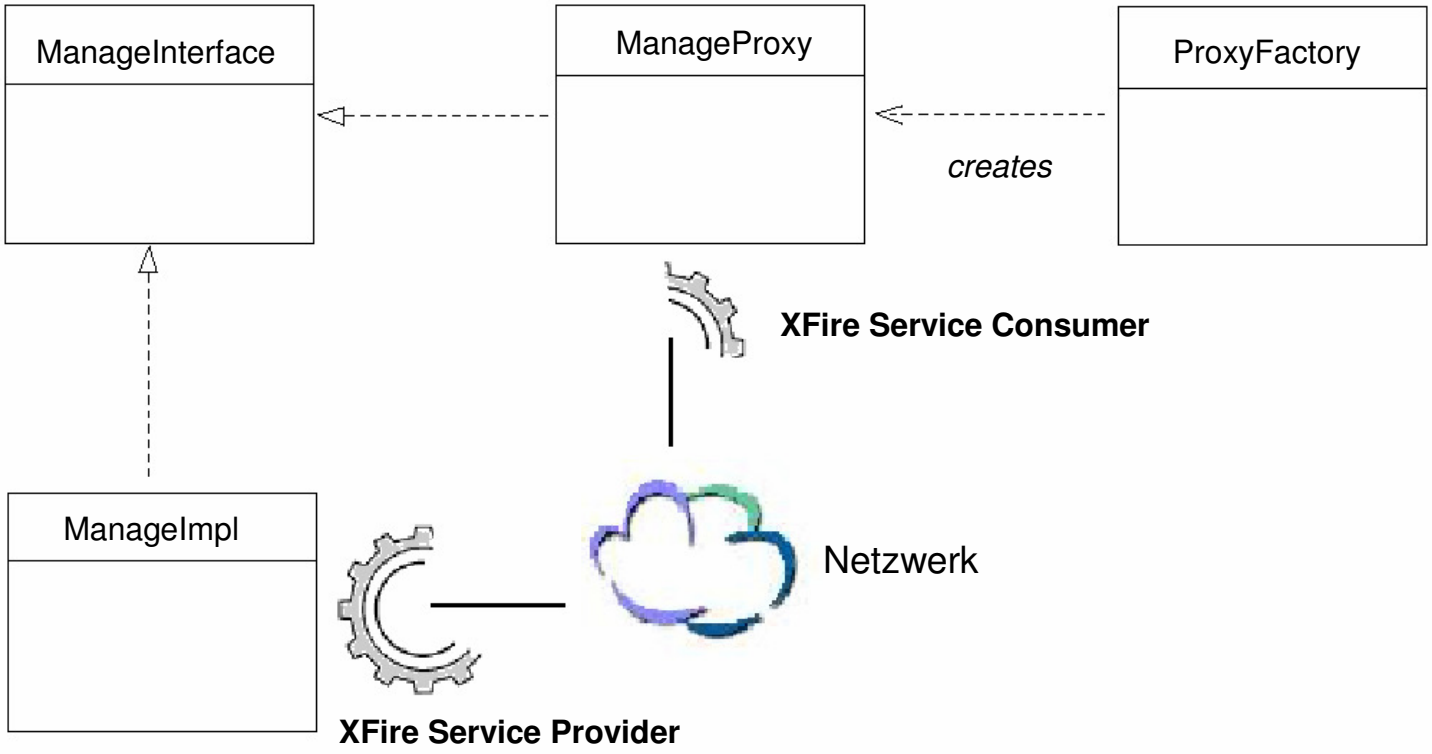
Praxisbeispiel 1 (Projekt: T-Systems/Daimler)

- mehrere Instanzen einer J EE Anwendung
- Shared Code
- Ziel: zentrales Vertrags-Management



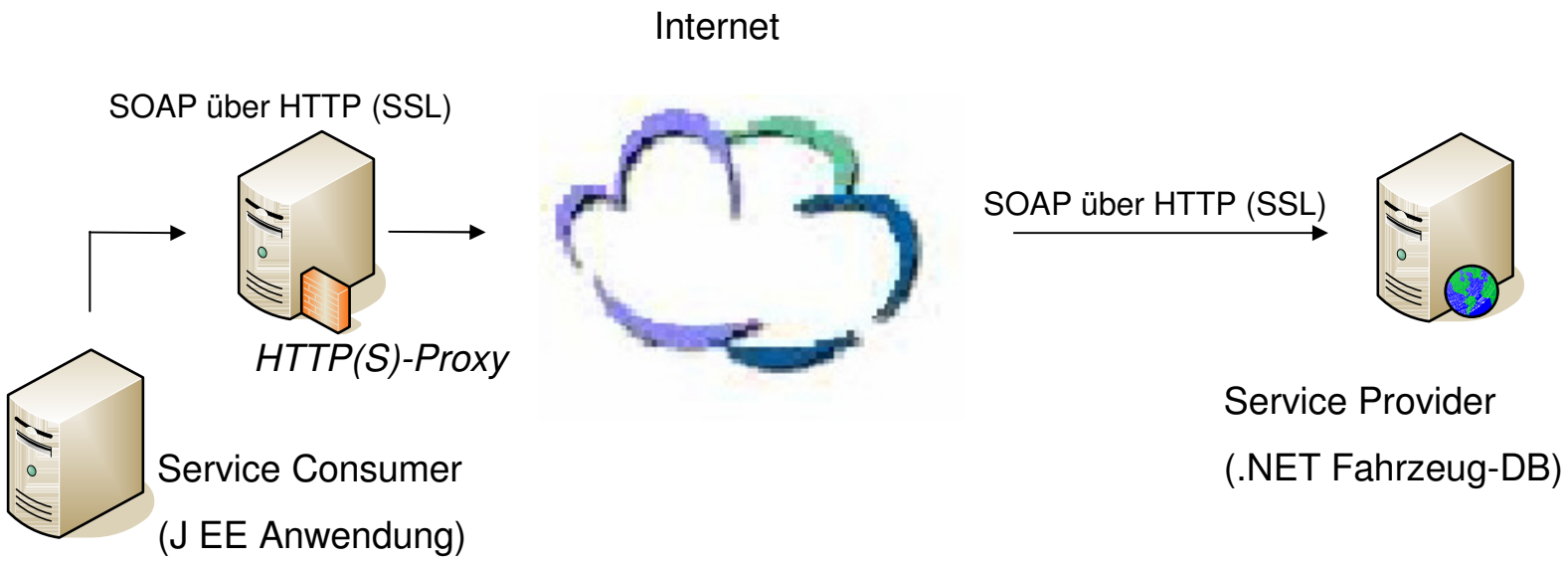
Praxisbeispiel 1 (Projekt: T-Systems/Daimler)

UML-ähnliche Darstellung der Schnittstelle



Praxisbeispiel 2 (Projekt: T-Systems/Daimler)

- Zugriff auf externe Fahrzeug-Datenbank
  - .NET basierter Web Service der externen Fahrzeug-DB
  - (Client-) Code-Generierung aus WSDL
  - HTTP(S)-Proxy, SSL, SOAP-Header-Authentifzierung



Fazit

- Vorteile
  - Einfache Verwendung besonders bezüglich
    - Code-Generierung
    - Service Providing durch XML-Deklaration
    - Service Consuming durch Client-Erzeugung aus Interface
  - wenig Refactoring notwendig
  - flexibel anpassbar durch austauschbare Module

- Nachteile
  - in der Praxis Probleme durch Standard-Abweichungen (generelles Problem bei Web Service Frameworks)
  - Dokumentation nicht ausführlich genug

Noch Fragen?

- Weiterführende Links / Artikel
  - http://xfire.codehaus.org/
  - http://eclipse-magazin.de/itr/online_artikel/psecom,id,864,nodeid,230.html
  - http://www.itarchitect.co.uk/articles/display.asp?id=344
  - http://www.javaworld.com/javaworld/jw-05-2006/jw-0501-xfire.html

Anhang: XFire - Verwendungsmöglichkeiten