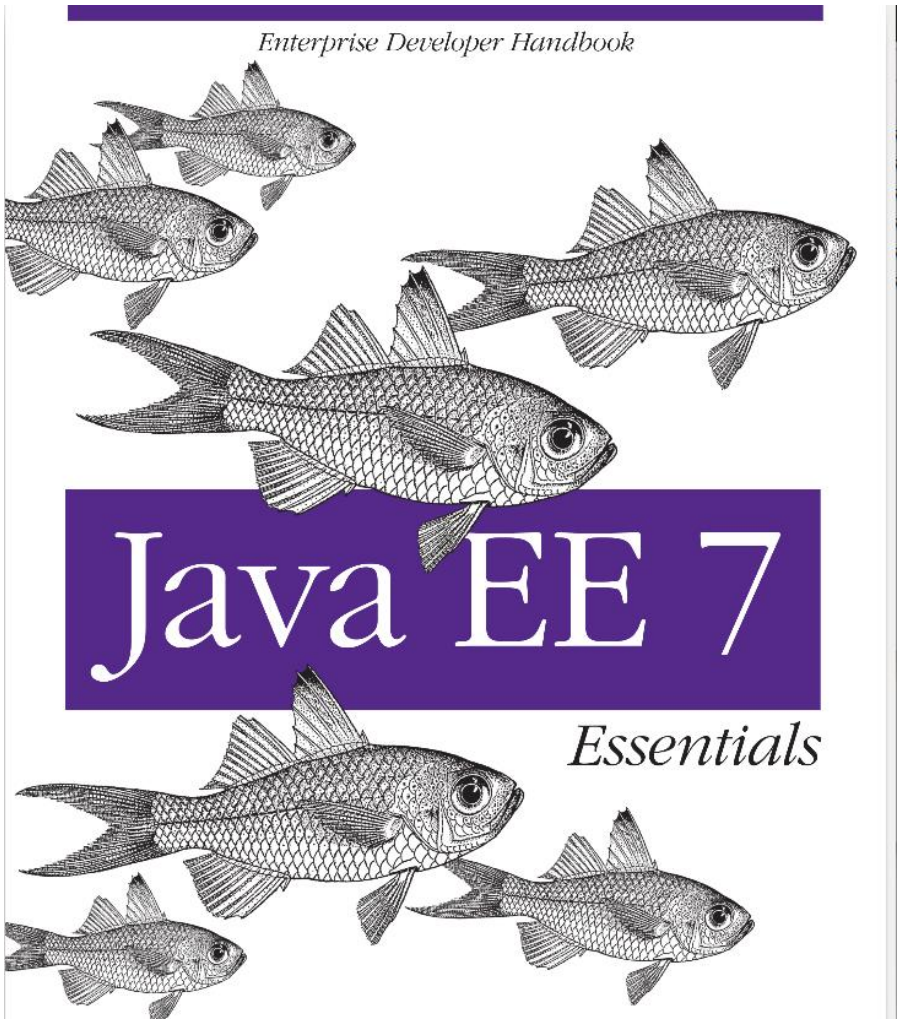
integrata
Cegos Group

inspire
qualify
change

SOAP-basierte Web Services

Literatur und Quellen



Einige Hinweise

integrata Cegos Group | inspire qualify change

- Die in diesem Seminar verwendete Werkzeuge und Frameworks sind Open Source
  - LPGL Lizenzmodell
- Dies ist ein Programmier-Seminar
  - Damit werden die Inhalte durch Übungen vertieft und verinnerlicht
  - Musterbeispiele werden zur Verfügung gestellt
  - Diese können am Ende des Seminars als ZIP-Datei kopiert werden
    - USB-Stick oder ähnliches
- Dokumentation und Ressourcen stehen auch im Internet zur Verfügung
- Konventionen
  - Befehle werden in `Courier-Schriftart` dargestellt
  - Dateinamen werden in *`kursiver Courier-Schriftart`* dargestellt
  - Links werden in `unterstrichener Courier-Schriftart` dargestellt
  - Zitate werden in *"Anführungszeichen kursiv"* formatiert, die Quellenangabe steht eingerückt darunter

Copyright und Impressum

Javacream

Dr. Rainer Sawitzki

Alois-Gilg-Weg 6
81373 München

Inhalt

integrata Cegos Group
inspire qualify change

1

MODERNE SOFTWARE-ARCHITEKTUREN

1.1

ANWENDUNGSWENTWICKUNG

Typische Anforderungen

integrata Cegos Group
inspire qualify change

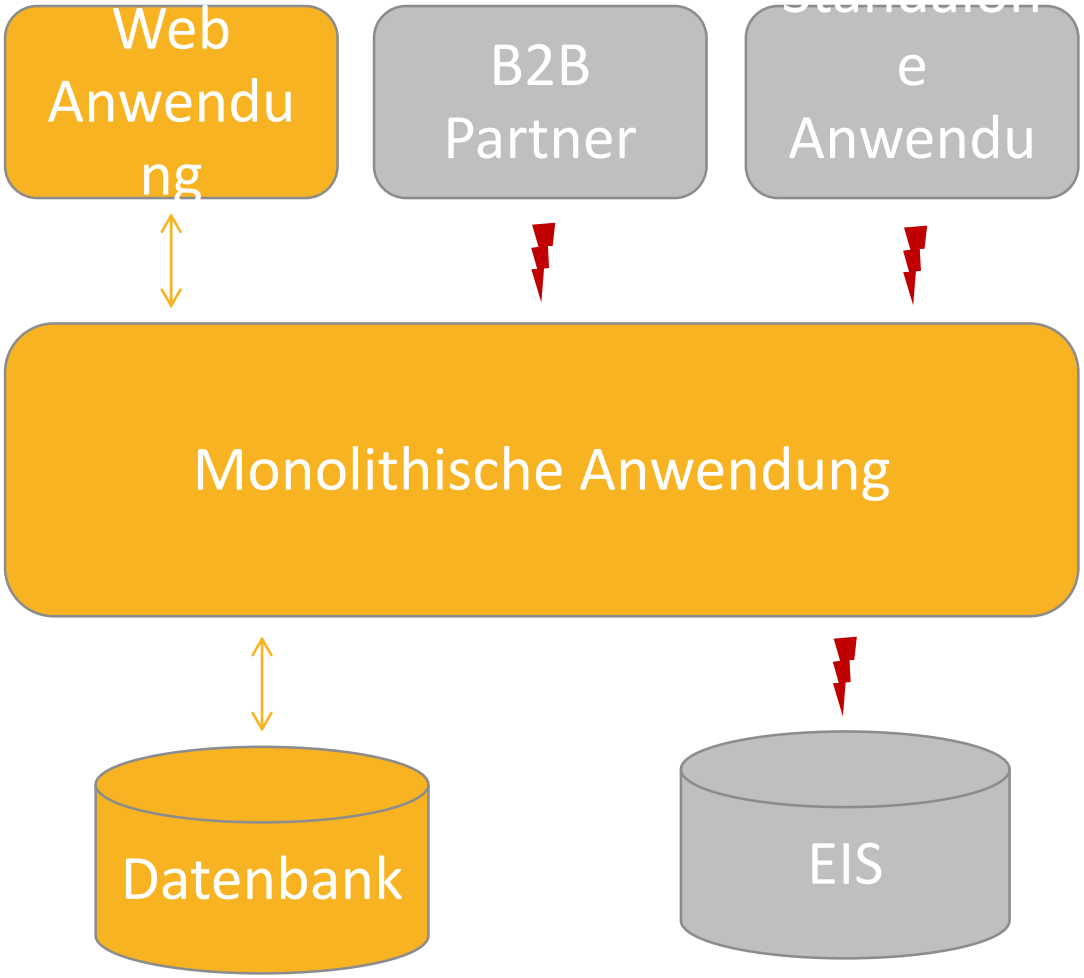
- Modularisierung und Schichten-Architektur
  - Komplexere Anwendungen können somit nur durch die Kollaboration vieler relativ simpler Objekte realisiert werden, die ein komplex verschachteltes Objekt-Geflecht bilden.
- Resource Management
  - Zugriff auf Datenbanken und andere Enterprise Information Systeme
  - Die Anwendung enthält Zugriffs-Logik und das Transaktionsmanagement
- Nutzen zentraler Dienste
  - Logging
  - Security
  - …
- Test
- Remoting
  - Implementierungen können auf verschiedenen Systemen verteilt werden

Heterogenität und Komplexität

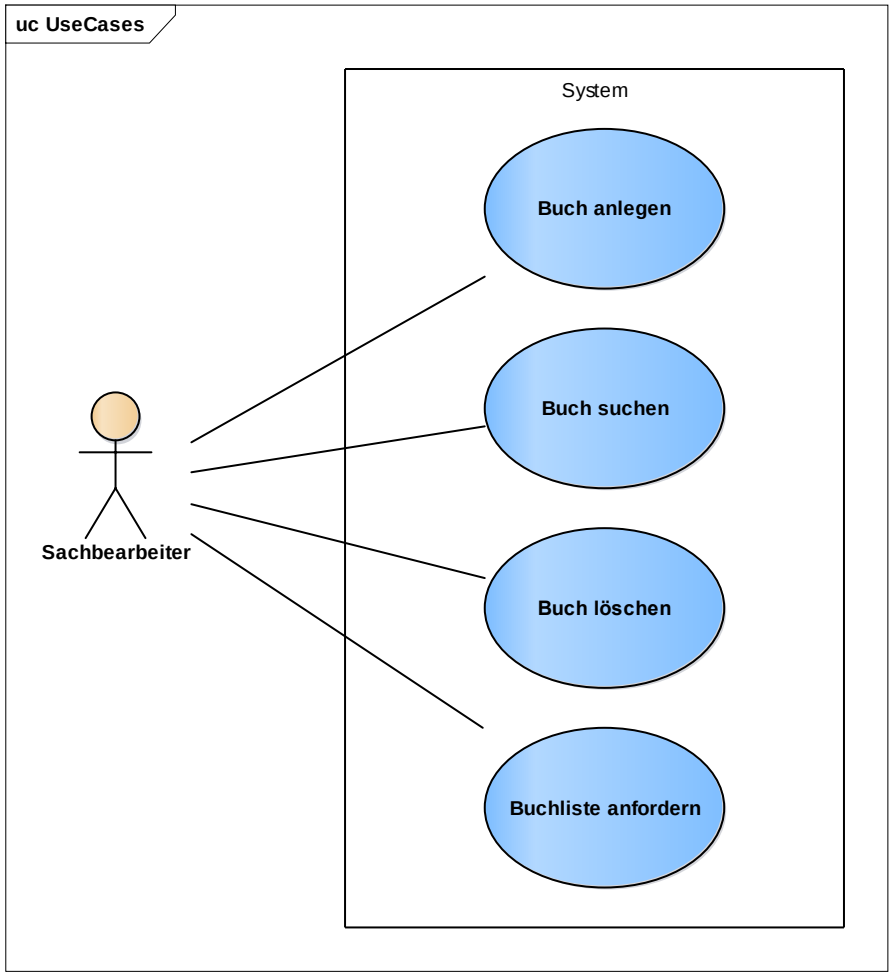
- Ein Unternehmen benutzt/enthält
  - Eine Vielzahl verschiedenster Client-Technologien und verwendeter Kommunikationsprotokolle, die einem steten Wandel unterzogen sind
  - Eine Vielzahl vorhandener Enterprise Ressourcen mit unterschiedlichen Backend-Technologien, die aus Unternehmenspolitischen und/oder technischen Gründen ebenfalls jederzeit Wechseln können
  - Eine Menge allgemeiner Unternehmens-spezifischer Abläufe und Prozesse, die mit den verschiedensten Programmiersprachen und Werkzeugen ausgeführt werden

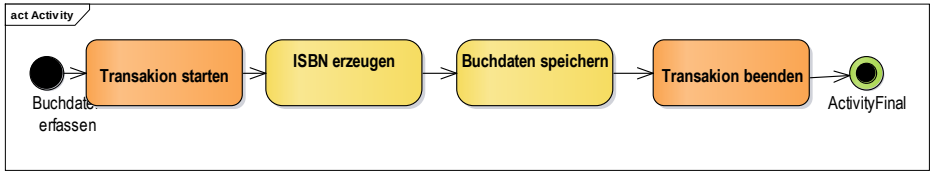
Eingeschränkte Wiederverwendbarkeit

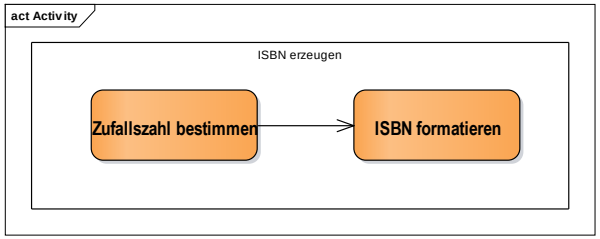
integrata Cegos Group
inspire qualify change



Eine einfache Vorgabe





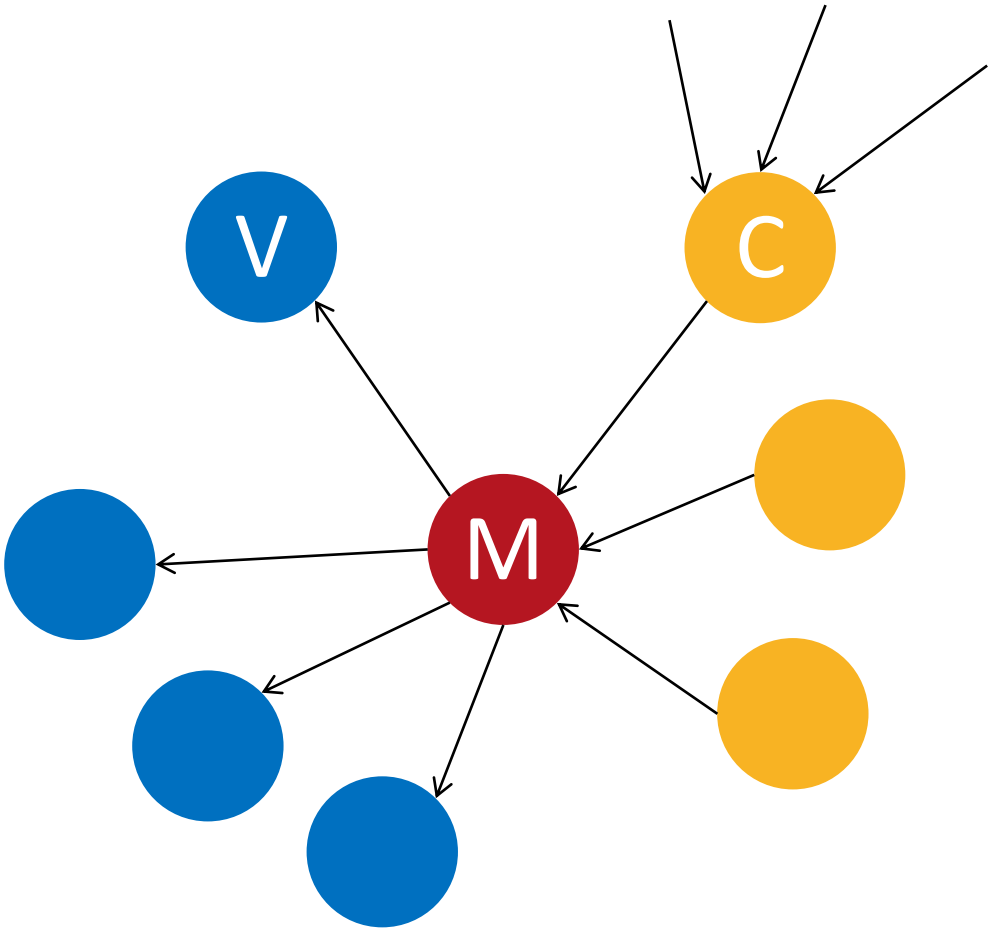


1.2

MODELLIERUNG

Design

- Monolithisches Design
  - Logik wird in einem zentralen Programm komplett abgelegt
- Objektorientiertes Design
  - Modularisierung durch Rollen-Konzept
- Pattern-orientiertes Design
  - Umsetzung der Applikation durch Anwendung etablierter Entwurfsmuster
    - Beispiel: Model View Controller (MVC)


V
C
M

Service-orientierte Architekturen

integrata Cegos Group
inspire qualify change

- Service-orientierte Architekturen (SOA)
  - Wohldefinierte Schnittstellenbeschreibung
  - Plattform-unabhängige Installation
  - Zustandslose Implementierung
  - Ansprechen über ein im Wesentlichen frei wählbares Kommunikationsprotokoll
  - Verwendung einer Registry

1.3

TECHNOLOGIEN

Übersicht

- Context und Dependecy Injection (CDI)
  - Ein Context übernimmt die Kontrolle über den Lebenszyklus aller Fachobjekte und erkennt und setzt deren Abhängigkeiten
    - Spring als bahnbrechendes Framework im Java-Umfeld
    - Mittlerweile im Java-Standard angekommen
- Querschnittsfunktionen mit Aspektorientierung
  - Auch bekannt unter Interceptor, Filter, „Cross Cutting Concerns“
  - Beispiele
    - Methoden-Tracing
    - Performance-Tests
    - Auditing
    - Transaktionssteuerung
    - Authentifizierung
    - ...

Eine Beispiel-Anwendung

integrata
Cegos Group
inspire
qualify
change

2

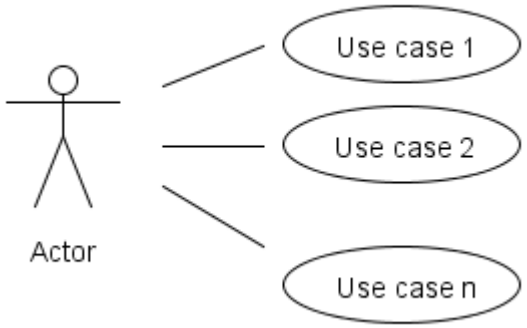
EINE ÜBERSICHT

2.1

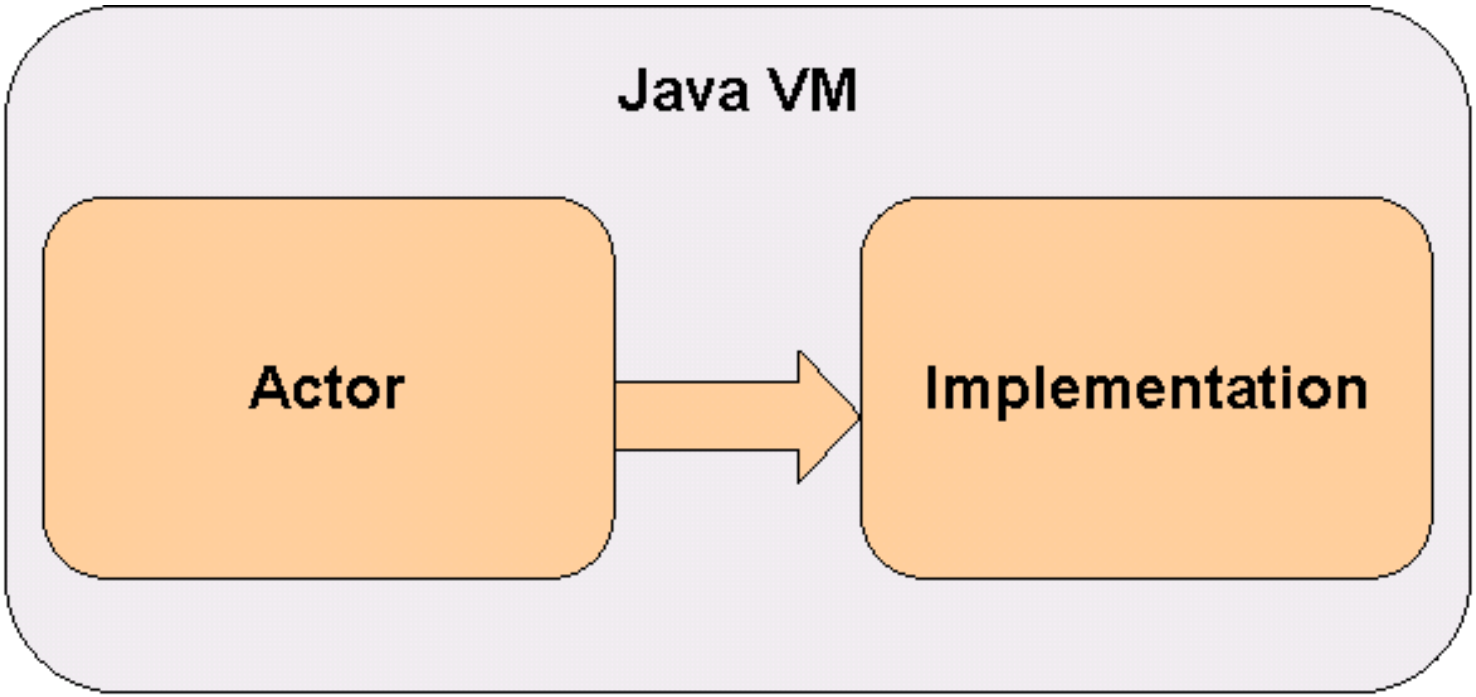
VON DER LOKALEN ANWENDUNG ZUM WEB SERVICE

Fachmodell

- Auch eine Web Services Anwendung wird ausschließlich Funktionalitäten umsetzen können, die beispielsweise durch eine klassische Use-Case-Analyse identifiziert werden
  - Web Services treten in einer fachlichen Modellierung nicht notwendigerweise auf

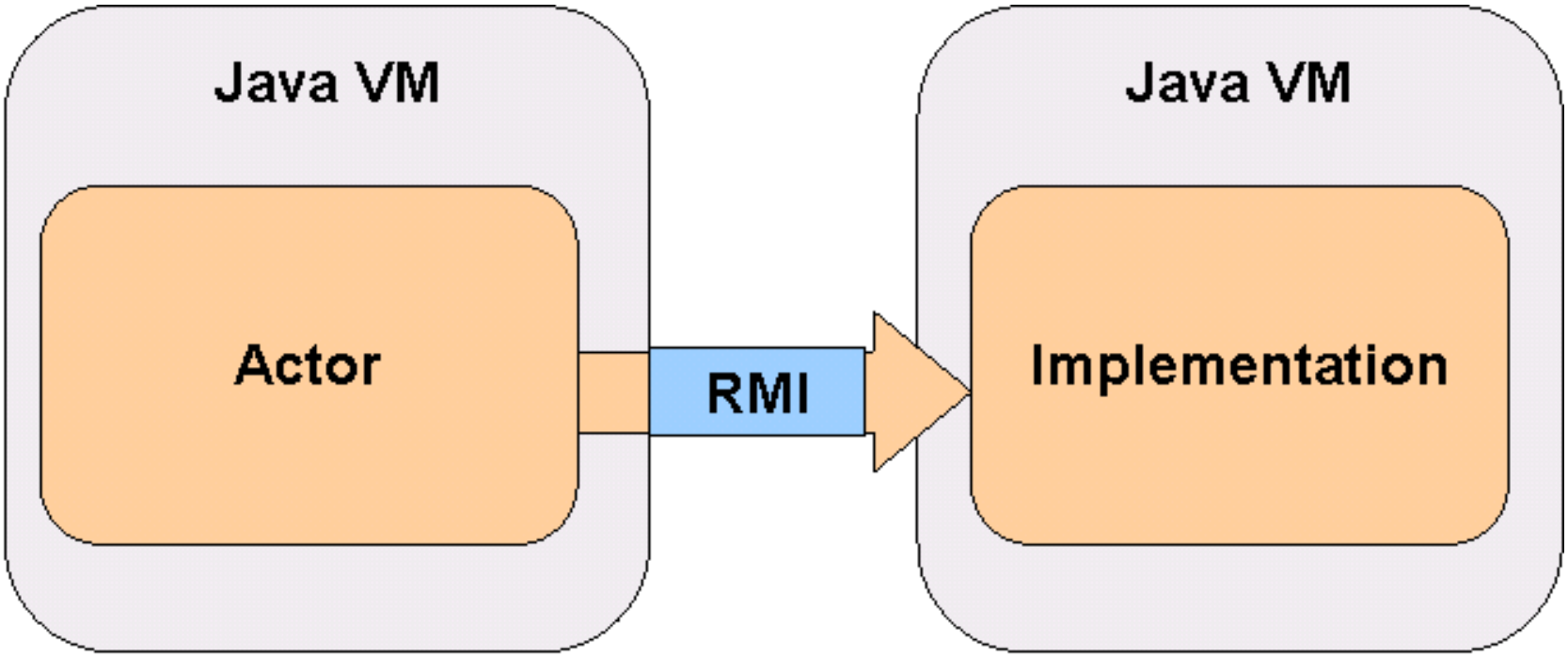


Lokal

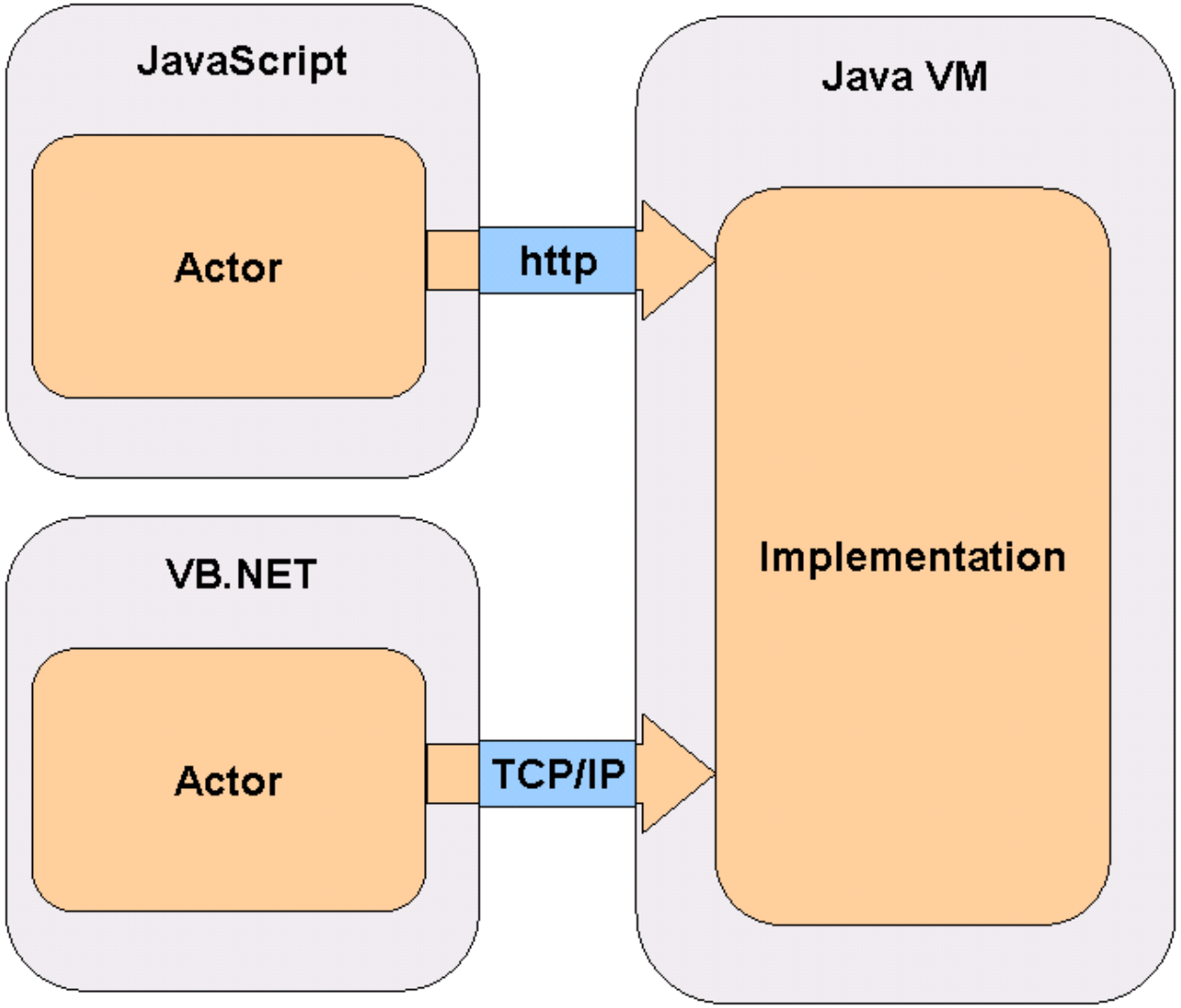


Verteilung mit Java RMI

integrata Cegos Group
inspire qualify change



Interoperabilität



2.2

EINFÜHRUNG WEB SERVICES

Bestandteile

- Web Services definieren primär ein Plattformunabhängiges Daten-Austauschformat sowie dessen Transport über ein Netzwerk-Protokoll
  - Zusätzlich können die Interfaces in einer Sprachunabhängigen, einfach auswertbaren Form programmiert werden
- Als Format wird bei Web Services XML benutzt:
  - Das Web Services Description Language-Schema (WSDL) erlaubt die Definition von Schnittstellen
  - Das SOAP-Envelope-Schema beschreibt das Datenaustausch-Format.

Vorsicht!

- Wichtig für die Bewertung dieses Ansatzes ist, sich hier noch mal die Problemstellung klar zu machen: Es geht hier um die Notwendigkeit, eine Service-Beschreibung in
  - vollständiger,
  - Sprach-unabhängiger,
  - einfach automatisiert verarbeitbarer und
  - dokumentierter Form zu erstellen
- Die hier vorgestellten Web Services sind aber gar keine echten „Web Services“!
  - Sie werden von einem Browser direkt nicht benutzt werden können
    - Dafür werden RESTful Web Services benutzt, die mit WSDL und SOAP nicht das Geringste zu tun haben

WSDL/SOAP versus REST

- SOAP Web Services fokussieren stark auf den Service-Ansatz
  - und werden deshalb vorwiegend zur Realisierung eher interner, komplexer Geschäftsprozesse benutzt
- Geht es „nur“ darum, einen entfernten Browser mit Informationen zu bedienen, sind RESTful Web Services deutlich besser geeignet
  - RESTful Web Services sind damit die eigentlichen Web Services

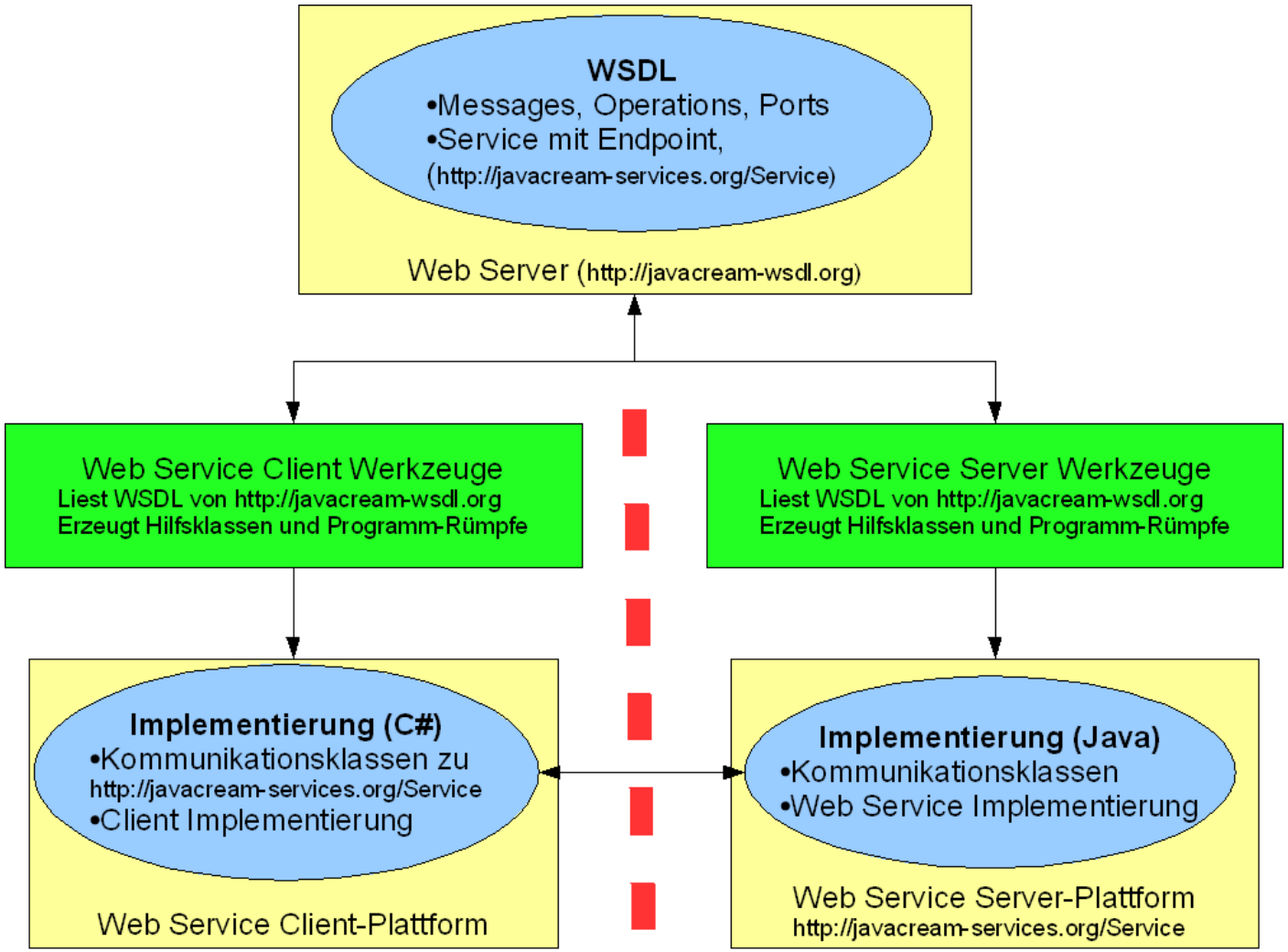
2.3

WSDL UND SOAP-ENVELOPE

Die WSDL

- Ein Web Service wird durch ein XML-Dokument beschrieben, dessen Schema durch die „Web Service Description Language“ definiert ist
  - Man spricht kurz von einer „WSDL“
- Eine WSDL-Datei beschreibt:
  - die einzelnen Service-Operationen,
  - die verwendeten Datentypen, die zwischen dem Web Service Client und der Web Service Implementierung ausgetauscht werden können,
  - das zur Kommunikation verwendete Protokoll,
  - die URL des Endpoints der Service-Implementierung

WSDL und Programmiersprachen



Der SOAP-Envelope zum Datenaustausch

integrata Cegos Group
inspire qualify change

2.4

THE DARK SIDE OF WEB SERVICES

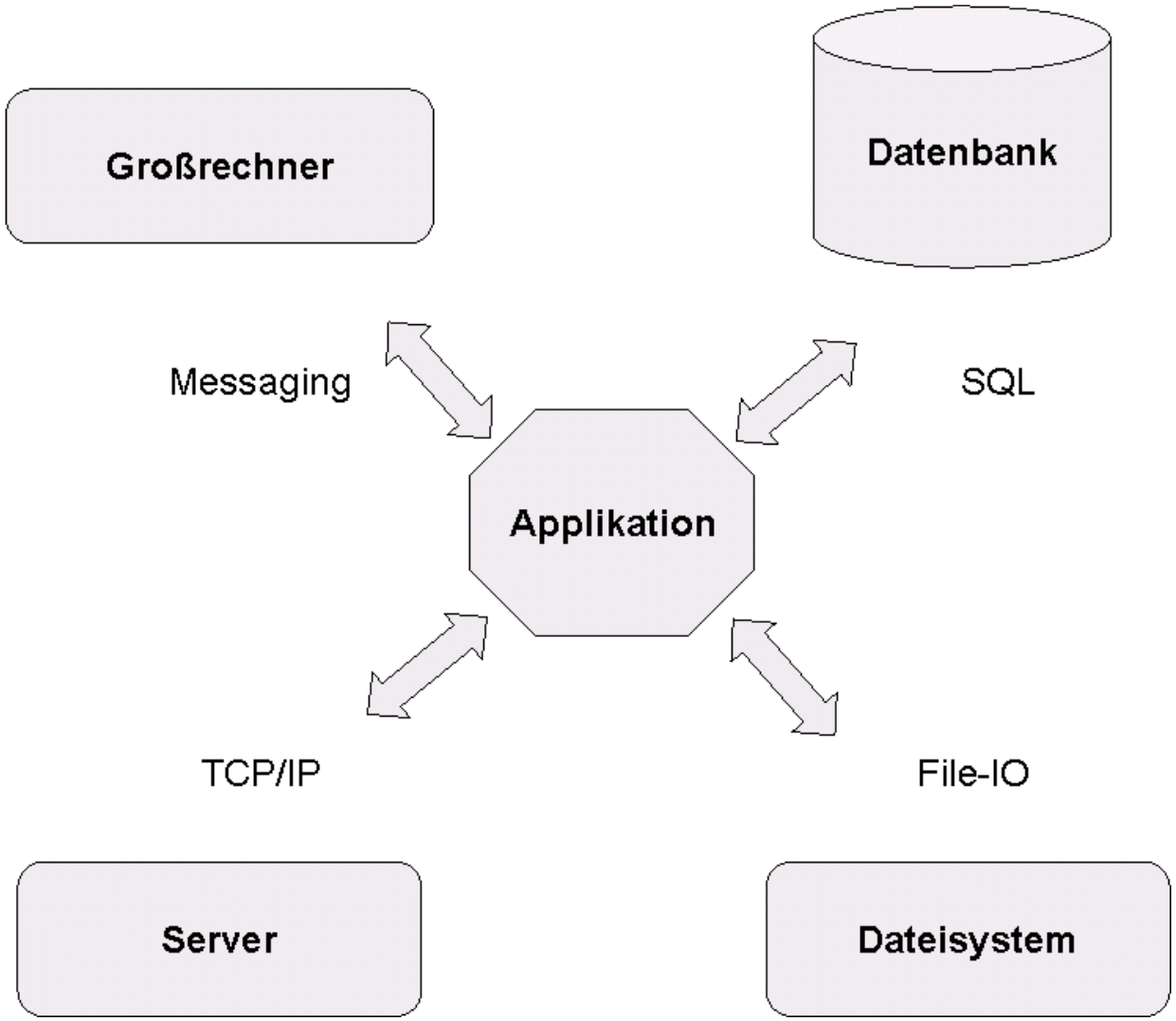
Grenzen der Spezifikation

integrata Cegos Group
inspire qualify change

- Verschiedene Styles
  - RPC/literal
  - RPC/encoded
  - Document/literal
  - Document/wrapped
- Wie wird die aufgerufene Operation übertragen?
  - Als Header-Property des SOAP-Envelopes?
  - Durch einen http-Header?
- Ein ganzer Zoo von optionalen WS-*-Erweiterungen
  - WS Security
  - WS Addressing
  - WS Transactions
  - …
  - Diese Erweiterungen sind für eine Web Services Plattform nicht verpflichtend!

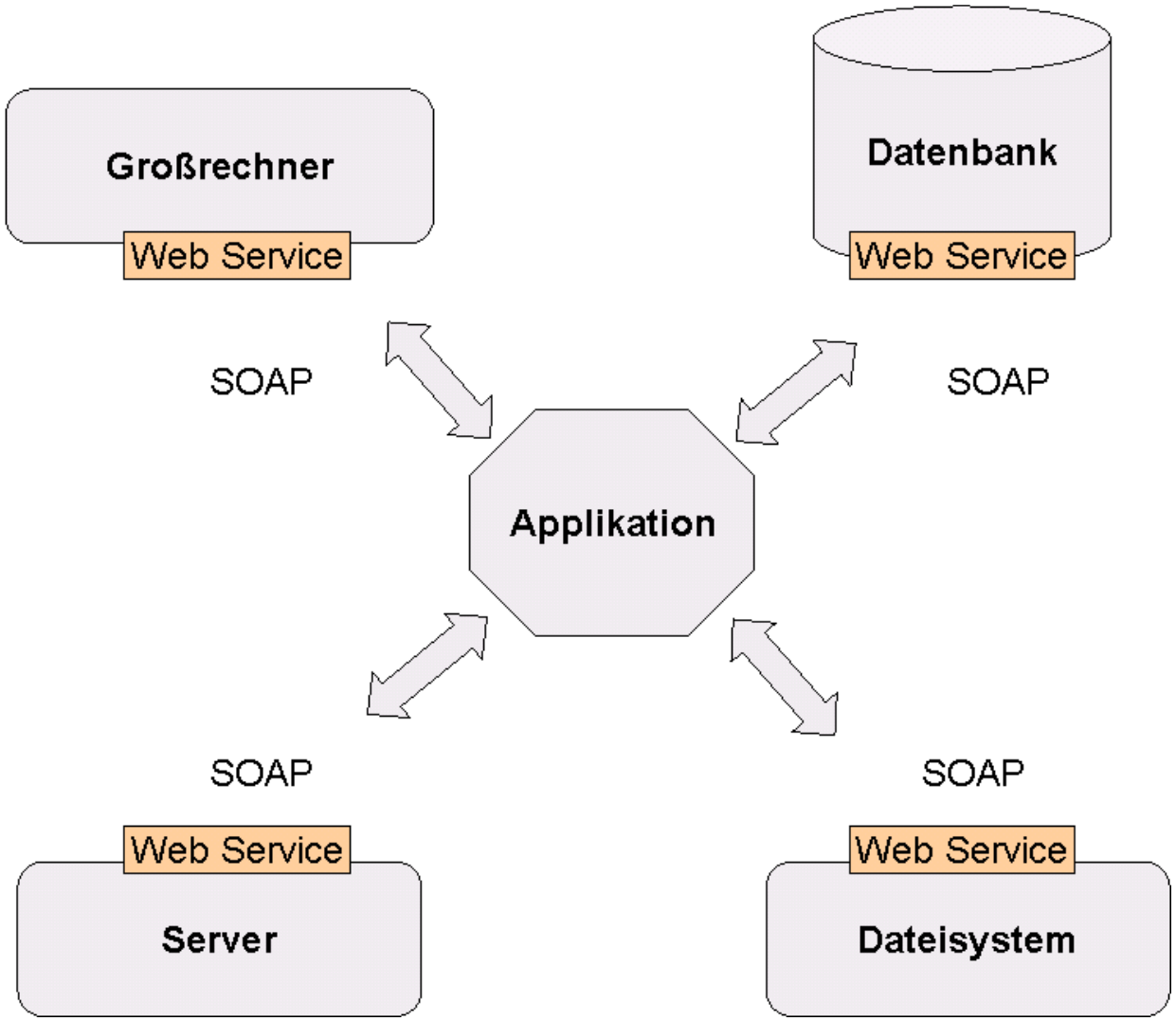
Eine funktionierende Altanwendung

integrata Cegos Group
inspire qualify change



Funktioniert die Umstellung auf Web Services?

integrata Cegos Group
inspire qualify change



Funktioniert die Umstellung auf Web Services?

- Eine schlichte Vereinfachung durch Einführung einer einheitlichen Web Services-Zugriffsschicht mag homogener aussehen, wird aber mit hoher Wahrscheinlichkeit eine völlig unbefriedigende Performance aufweisen
  - In der Praxis sind insbesondere in den Zeiten der SOA-Euphorie einige Projekte trotz immensem Aufwand niemals in den Produktionsbetrieb gegangen!
- Die Einführung und Verwendung von Web Services verlangt ein hohes technisches Verständnis sowie die Kenntnis der vorhandenen Beschränkungen und Risiken

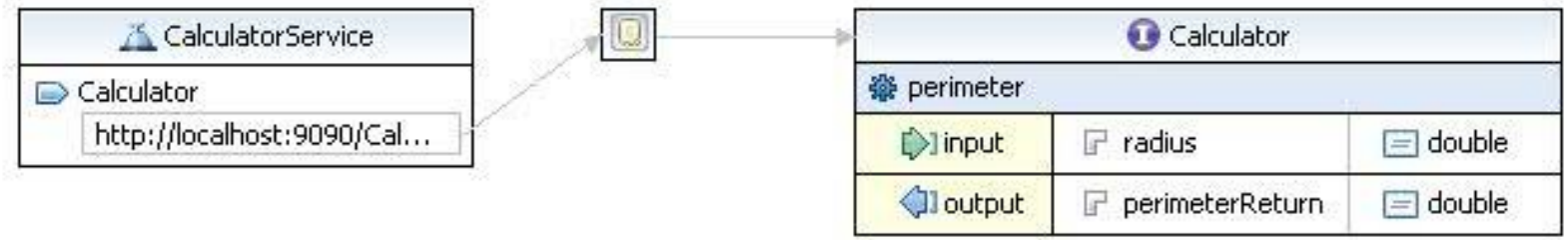
3

EINE ERSTE ANWENDUNG

3.1

ERSTE SCHRITTE

- Eine WSDL beschreibt analog einer simplen Java-Schnittstelle die Signatur eines Services
  - Beispiel: Eine WSDL im Eclipse-Editor



Der SOAP-Envelope

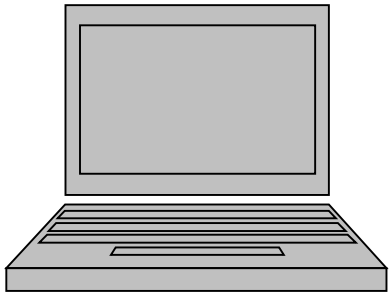
3.2

EINE BASIS-IMPLEMENTIERUNG IN JAVA

Eine minimale WebService-Implementierung

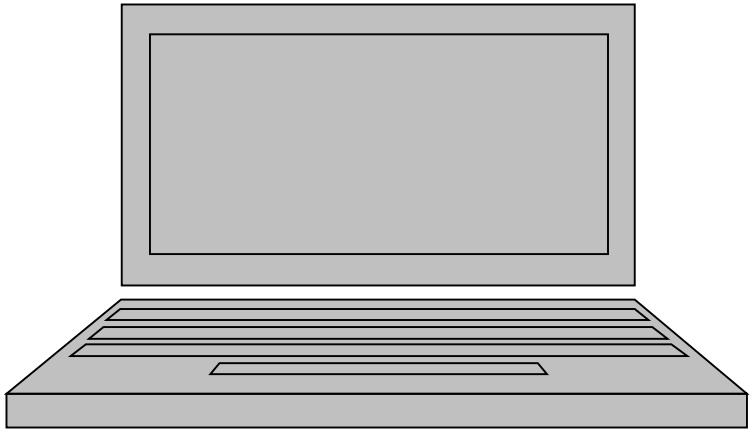
integrata Cegos Group
inspire qualify change

- Web Services benutzen mit XML-Verarbeitung und dem http-Protokoll Standard-Technologien
  - Deshalb kann prinzipiell sowohl ein Client als auch eine Server-Implementierung mit wenigen Zusatzbibliotheken realisiert werden.
- Client
  - URL Connection öffnen
  - Http-Connection konfigurieren
  - XML senden
  - XML empfangen
  - Verbindung schließen



Server

- Auch der Server funktioniert im Prinzip auf folgende Weise:
  - HttpRequest entgegen nehmen
  - Lesen von XML aus HttpRequest
  - Verarbeitung anstoßen
  - Schreiben des Ergebnisses in den HttpResponse



3.3

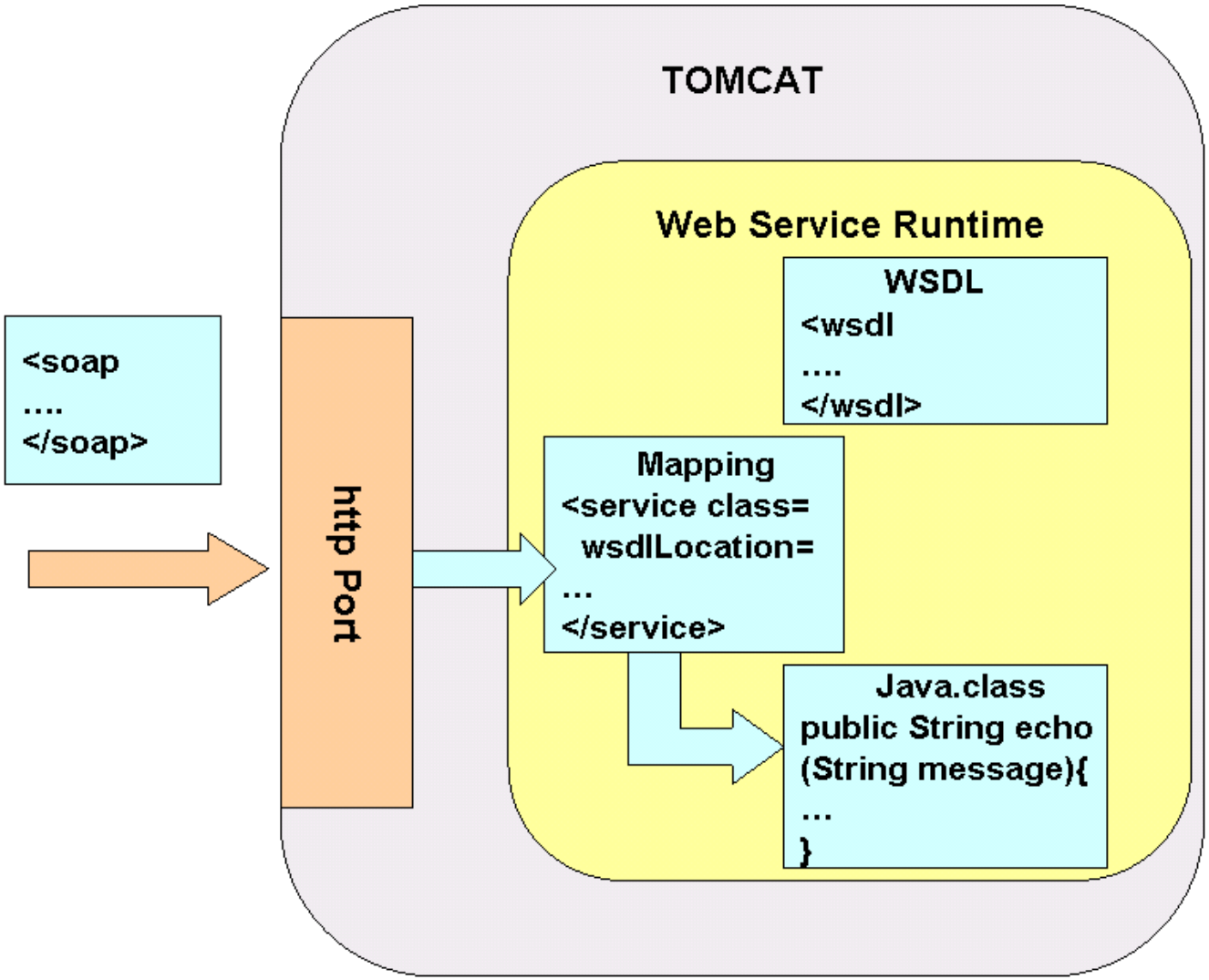
WEB SERVICES PROVIDER

Übersicht

- Benutzung von Konfigurations- bzw. Mapping-Dateien
  - Die in Mapping-Dateien enthaltenen Informationen werden von der Laufzeitumgebung genutzt, um einen Web Service Aufruf auf eine Java-Methode abzubilden
  - Die implementierende Java-Klasse ist in der Regel vollkommen unabhängig von Web Services Details
  - Eine Auswahl von Frameworks, die nach diesem Prinzip arbeiten, ist:
    - Apache Axis 1.x
    - Apache Axis 2
    - Apache CXF
    - Spring Web Services
- Annotationen
  - Beispiel JAX-WS als Bestandteil der Java-Standard-Bibliothek

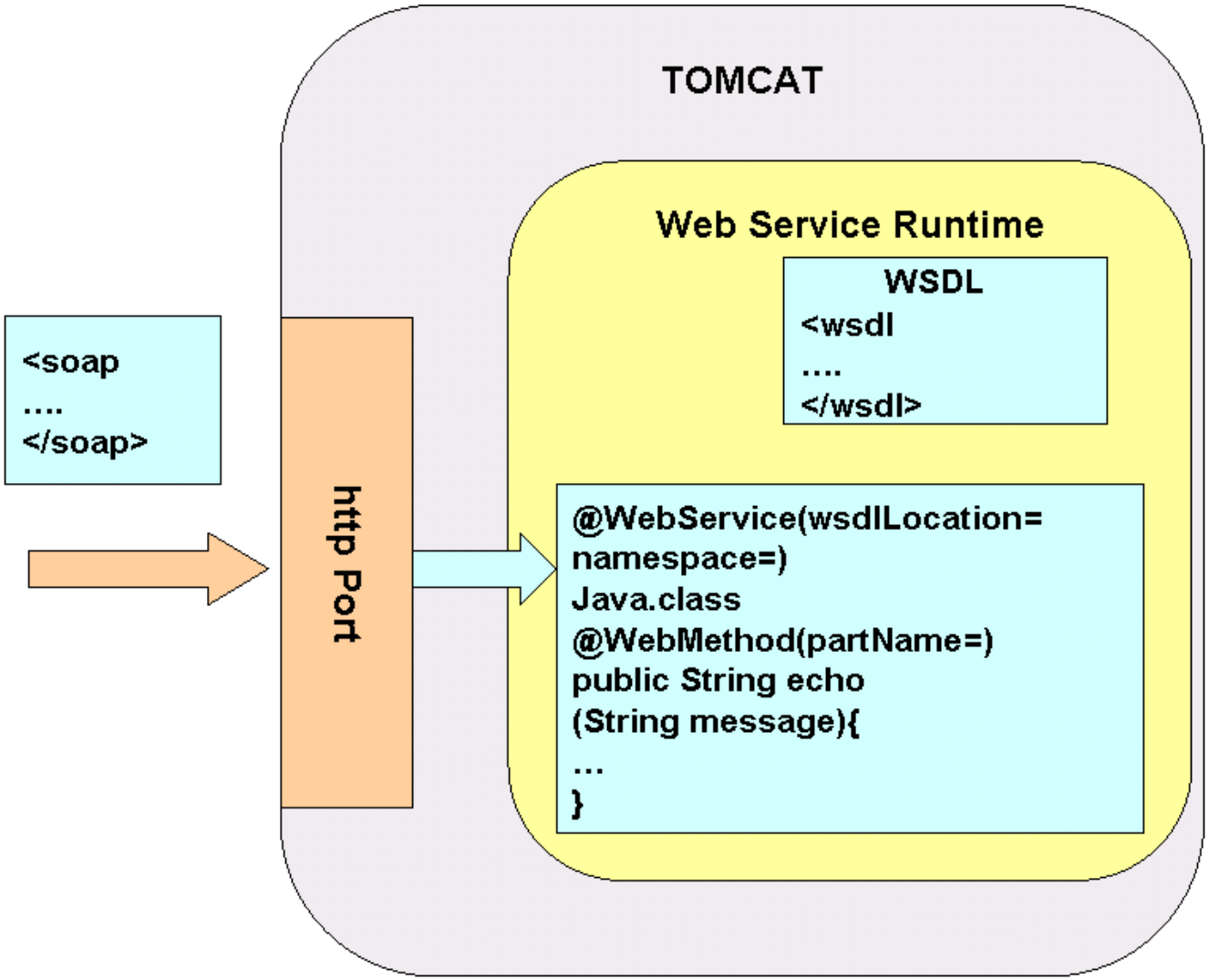
Konfigurativ

integrata Cegos Group
inspire qualify change



Annotationen

integrata Cegos Group
inspire qualify change



Historische Web Services
Laufzeitumgebungen

- Die Entwicklung hin zur JAX-WS-Spezifikation verlief nicht unbedingt geradlinig
- Betrachtet man das Gemenge der verschiedenen Java-Packages, die dem Web Services-Umfeld zugeordnet sind, so finden sich hier auch einige kaum noch benutzte bzw. tote Enden der Entwicklung.
  - JAX-RPC (Java API for XML-based RPC) war der Vorläufer von JAX-WS
    - Die Klassen und Schnittstellen des Pakets `javax.xml.rpc` sind deprecated.
  - JSXR (Java API for XML Registries) definiert ein API für den Zugriff auf XML-Registries
    - Diese Registries spielen in der Praxis jedoch keine große Rolle

4

DIE WSDL IM DETAIL

4.1

DAS WSDL-SCHEMA

WSDL-Element

integrata Cegos Group | inspire qualify change

- Die WSDL selbst ist ein einfaches XML-Dokument mit Verweis auf das zugehörige XML-Schema

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<wsdl:definitions
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:tns="http://www.example.org/IsbnGenerator/"
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  name="IsbnGenerator"
  targetNamespace="http://www.example.org/IsbnGenerator/">
```

- Type-Definitionen
  - XML-Schema mit XML-Standard-Datentypen
  - Einfache XSD-Typen (double, string)
  - Complex-Types = Datencontainer
  - Importieren externer Schema-Definitionen

Type-Definitionen

- XML-Schema mit XML-Standard-Datentypen
- Einfache XSD-Typen (double, string)
- Complex-Types = Datencontainer
- Importieren externer Schema-Definitionen

Messages

- Type-Deklaration + optional Part-Name im SOAP-Request.

```
<wsdl:message name="nextIsbnRequest">
    <wsdl:part element="tns:nextIsbn" name="parameters"/>
  </wsdl:message>
  <wsdl:message name="nextIsbnResponse">
    <wsdl:part element="tns:nextIsbnResponse"
  name="parameters"/>
  </wsdl:message>
```

Port Types und Operations

- Besteht aus:
  - Operations
  - Diese wiederum bestehen aus
    - Name
    - Input
    - Output
    - Fault

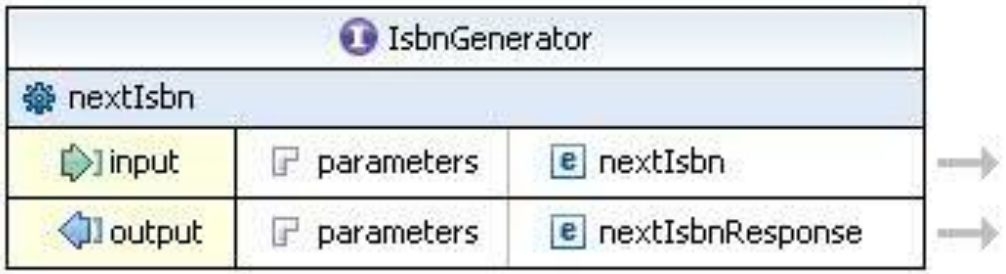
Analogien

integrata Cegos Group
inspire qualify change

XSD-Datentypen	Einfache Datentypen, `java.util.Date, TimeStamp, javax.xml.QName`
(Complex-)Types	Java-Klasse ausschließlich mit trivialen Methoden (getter/setter/Parameterloser Konstruktor, keine Business-Logik. Vorsicht: Contraints können in Java so nicht umgesetzt werden. Hierzu ist ein komplettes Validierungs-Framework notwendig!
Messages	Hier ist keine echte Analogie möglich, Messages entsprechen zentral definierten Parameterlisten bzw. Rückgabetypen.
Port-Type/Interface	Interface
Operations	Abstrakte Methode

Visualisierung im Eclipse Editor

integrata Cegos Group
inspire qualify change



Binding

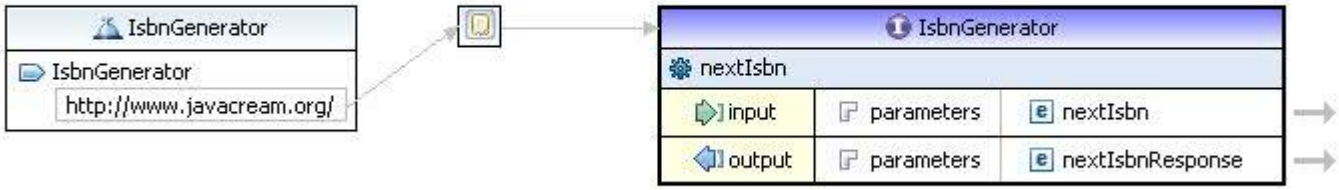

```
<wsdl:binding name="IsbnGeneratorSOAP„
type="tns:IsbnGenerator">
      <soap:binding style="document"

      transport="http://schemas.xmlsoap.org/soap/http"/>
      <wsdl:operation name="nextIsbn">
            <soap:operation
      soapAction="nextIsbn" />
            <wsdl:input>
                  <soap:body use="literal" />
            </wsdl:input>
            <wsdl:output>
                  <soap:body use="literal" />
            </wsdl:output>
      </wsdl:operation>
  </wsdl:binding>
```

Historie der SOAP-Bindings

- Ursprünglich sollten Web Services nichts anderes ermöglichen, als einfache Operationen via http-Protokoll aufrufen zu können
  - Damit war der RPC-Style geboren
- Besser war der Document-Style
  - Pro Operation darf hier nur ein einziger Parameter benutzt werden
- Diese Einschränkung wird durch den aktuellen Document/wrapped-Style wieder aufgehoben.

Service

```
<wsdl:service name="IsbnGenerator">
    <wsdl:port binding="tns:IsbnGeneratorSOAP"
name="IsbnGenerator">
      <soap:address
location="http://www.javacream.org/"/>
    </wsdl:port>
  </wsdl:service>
</wsdl:definitions>
```



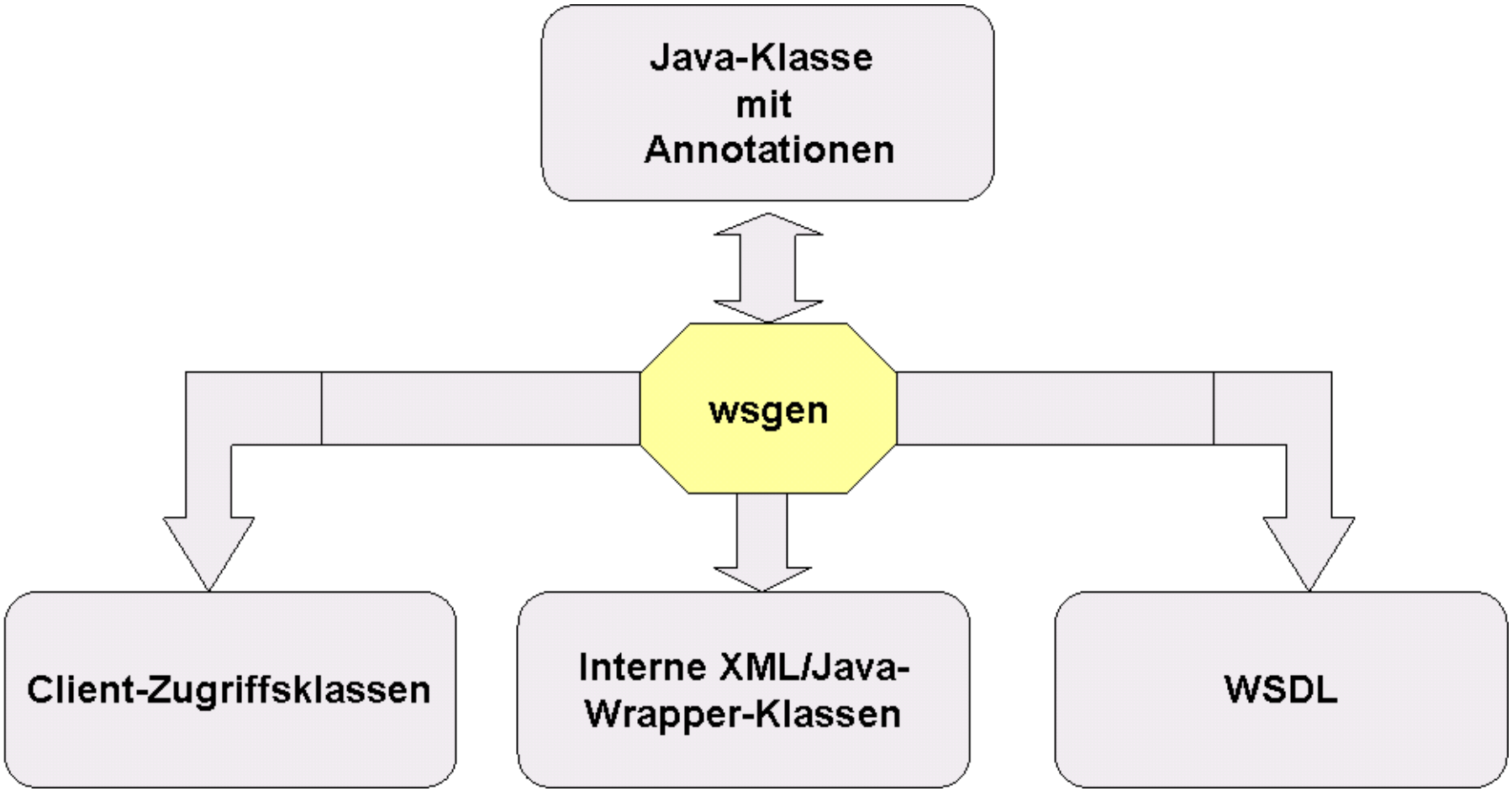
4.2

WSDL UND PROGRAMMIERSPRACHEN

WSDL und Java

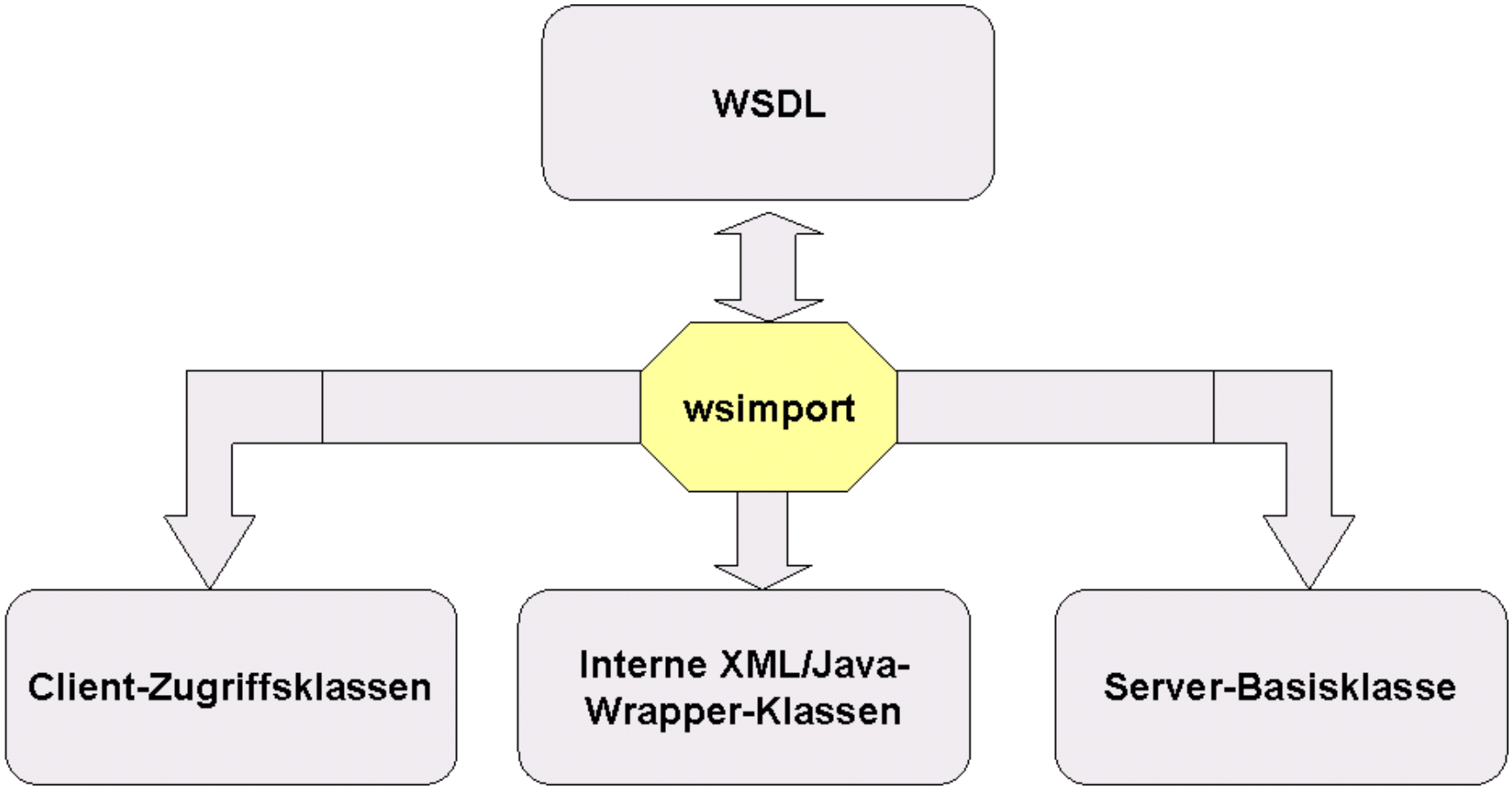
- WSDL und Java passen nicht nahtlos zusammen
  - Namensräume in XML sind URIs, Packages in Java haben eine andere Namens-Konvention
  - Methoden in Java-Klassen können überladen werden, Operationen nicht.
  - XML-Datenstrukturen definieren sehr einfach Constraints, dies ist in Java nur aufwändig möglich
  - Java-Klassen können selbstverständlich neben den trivialen Zugriffsmethoden auf Attribute komplexe Business-Logik enthalten. Dies kann keinesfalls nach XML umgesetzt werden
  - Vererbungs-Hierarchien können ebenfalls nicht nach XML umgesetzt werden

Erzeugung der WSDL aus Java



Erzeugung von Java-Programmrümpfen

integrata Cegos Group
inspire qualify change



5

REALISIERUNG EINES WEB SERVICES MIT JAVA

5.1

EINE SERVER-IMPLEMENTIERUNG MIT CODE FIRST

Realisierung eines Web Services

integrata Cegos Group | inspire qualify change

- Die in den folgenden Programmbeispielen verwendeten Typen sind Bestandteil der JAX-WS-Bibliothek mit den Packages
  - `javax.jws`
  - `javax.jws.soap`
  - `javax.xml.ws`
  - `javax.xml.ws.handler`
  - `javax.xml.ws.handler.soap`
  - `javax.xml.ws.http`
  - `javax.xml.ws.soap`
- Diese sind ab dem JDK 6 Bestandteil des JDK, vorher der Java Enterprise Edition (JEE)
- Die zugehörige Beschreibung ist nicht Bestandteil dieser Unterlage. Sie kann dem jeweiligen Javadoc entnommen werden

Code First

integrata Cegos Group | inspire qualify change

- Prinzipieller Ablauf
  - Beim Code First-Ansatz wird die WSDL als generiertes Artefakt aufgefasst
    - Der Entwickler des Web Services kümmert sich nicht um XML sondern bleibt ausschließlich im Java-Umfeld.
  - Es wird ein „Plain Old Java Object“, ein POJO programmiert
    - Einige Web Services Plattformen verlangen, dass das POJO eine Schnittstelle implementiert
    - Dies ist jedoch konzeptuell optional
- Das POJO wird mit Annotationen aus der JAX-WS-Bibliothek ergänzt

```
@WebService(name = "KeyGenerationWebService")
@SOAPBinding(parameterStyle = SOAPBinding.ParameterStyle.WRAPPED)
 public class RandomKeyGenerator{
```

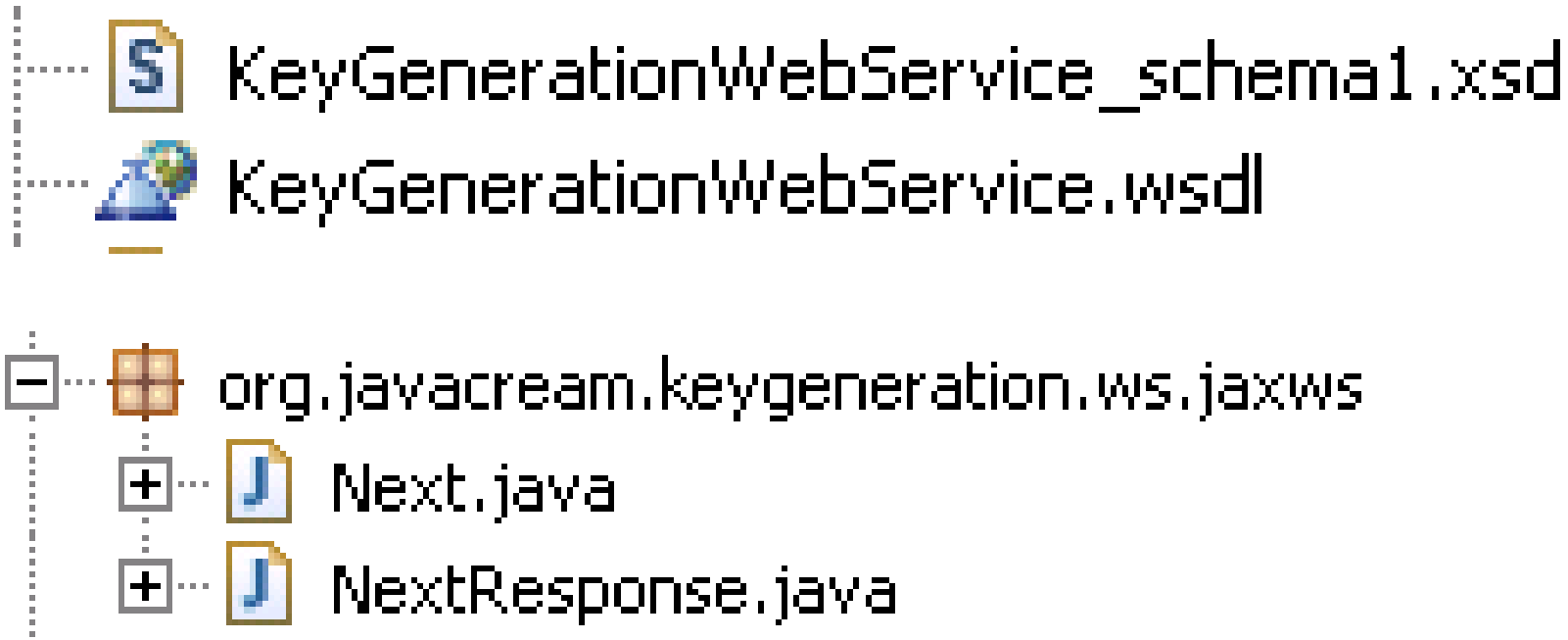
Benutzerdefinierte Datentypen

integrata Cegos Group | inspire qualify change

- Falls vorhanden werden die benutzerdefinierten Datentypen mit JAXB[1]-Annotationen versehen

```
@XmlAccessorType(XmlAccessType.FIELD)
@XmlType(name = "StoreEntry",
namespace="http://javacream.org/store/")
@XmlRootElement(namespace="http://javacream.org/store/",
name = "StoreEntry")
public class StoreEntry {
	@XmlElement(namespace="http://javacream.org/store/",
required = true)
	private String category;
	@XmlElement(namespace="http://javacream.org/store/",
required = true)
	private String item;
```

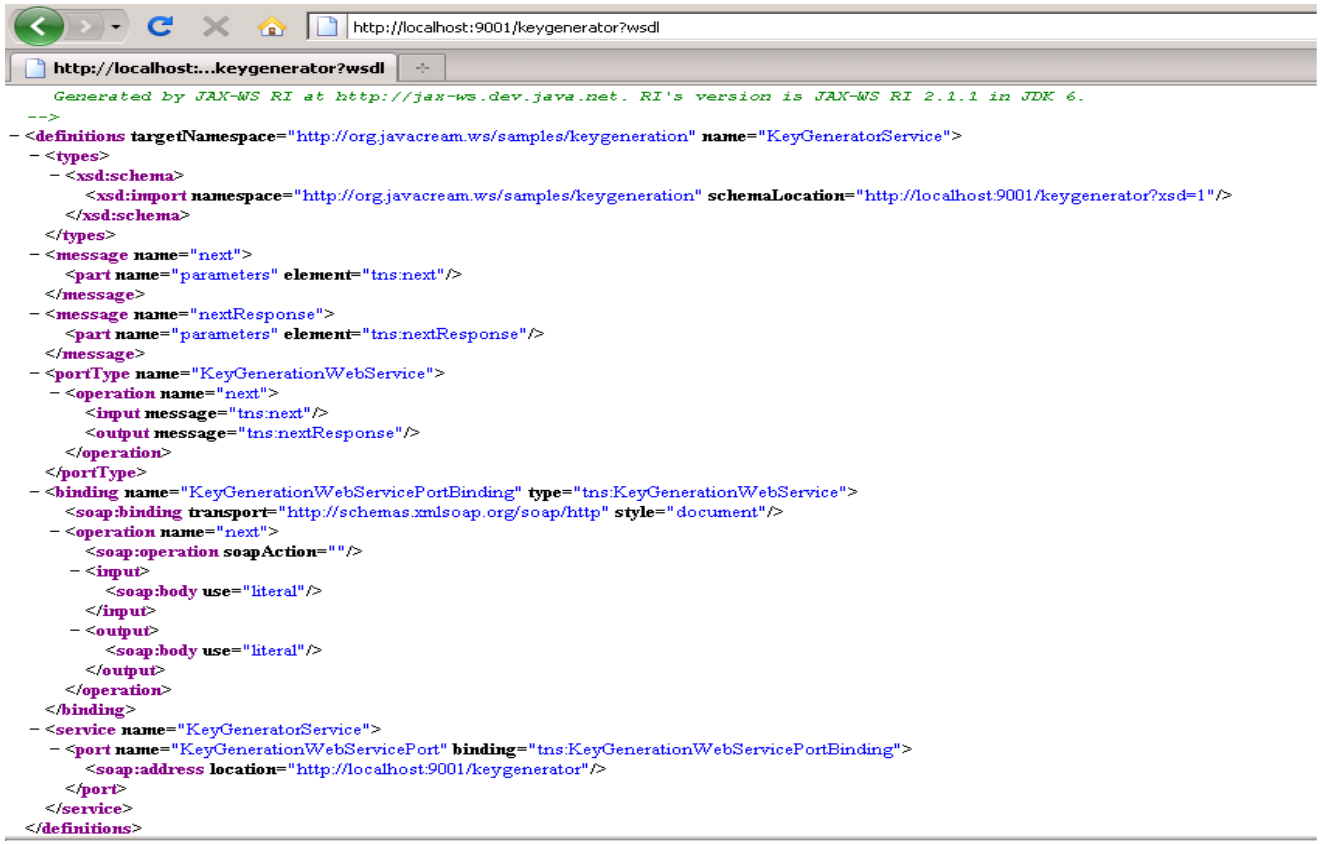
Die generierten Dateien



Der laufende Web Service

integrata Cegos Group | inspire qualify change

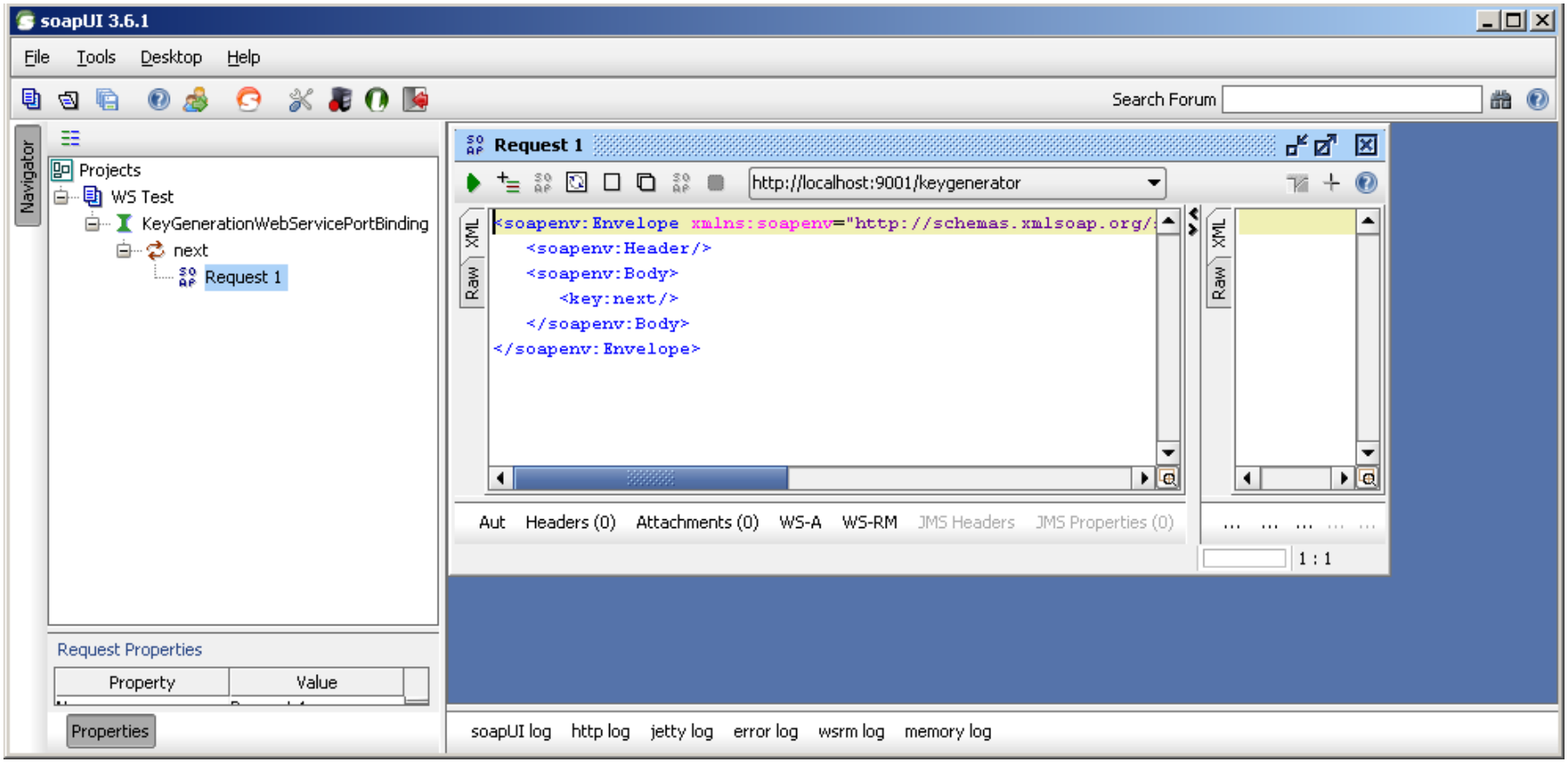
http://localhost:9091/keygenerator.wsdl




```
<!-- Generated by JAX-WS RI at http://jax-ws.dev.java.net. RI's version is JAX-WS RI 2.1.1 in JDK 6. -->
<definitions targetNamespace="http://org.javacream.ws/samples/keygeneration" name="KeyGeneratorService">
  <types>
    <xsd:schema>
      <xsd:import namespace="http://org.javacream.ws/samples/keygeneration" schemaLocation="http://localhost:9001/keygenerator?xsd=1"/>
    </xsd:schema>
  </types>
  <message name="next">
    <part name="parameters" element="tns:next"/>
  </message>
  <message name="nextResponse">
    <part name="parameters" element="tns:nextResponse"/>
  </message>
  <portType name="KeyGenerationWebService">
    <operation name="next">
      <input message="tns:next"/>
      <output message="tns:nextResponse"/>
    </operation>
  </portType>
  <binding name="KeyGenerationWebServicePortBinding" type="tns:KeyGenerationWebService">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document"/>
    <operation name="next">
      <soap:operation soapAction=""/>
      <input>
        <soap:body use="literal"/>
      </input>
      <output>
        <soap:body use="literal"/>
      </output>
    </operation>
  </binding>
  <service name="KeyGeneratorService">
    <port name="KeyGenerationWebServicePort" binding="tns:KeyGenerationWebServicePortBinding">
      <soap:address location="http://localhost:9001/keygenerator"/>
    </port>
  </service>
</definitions>
```

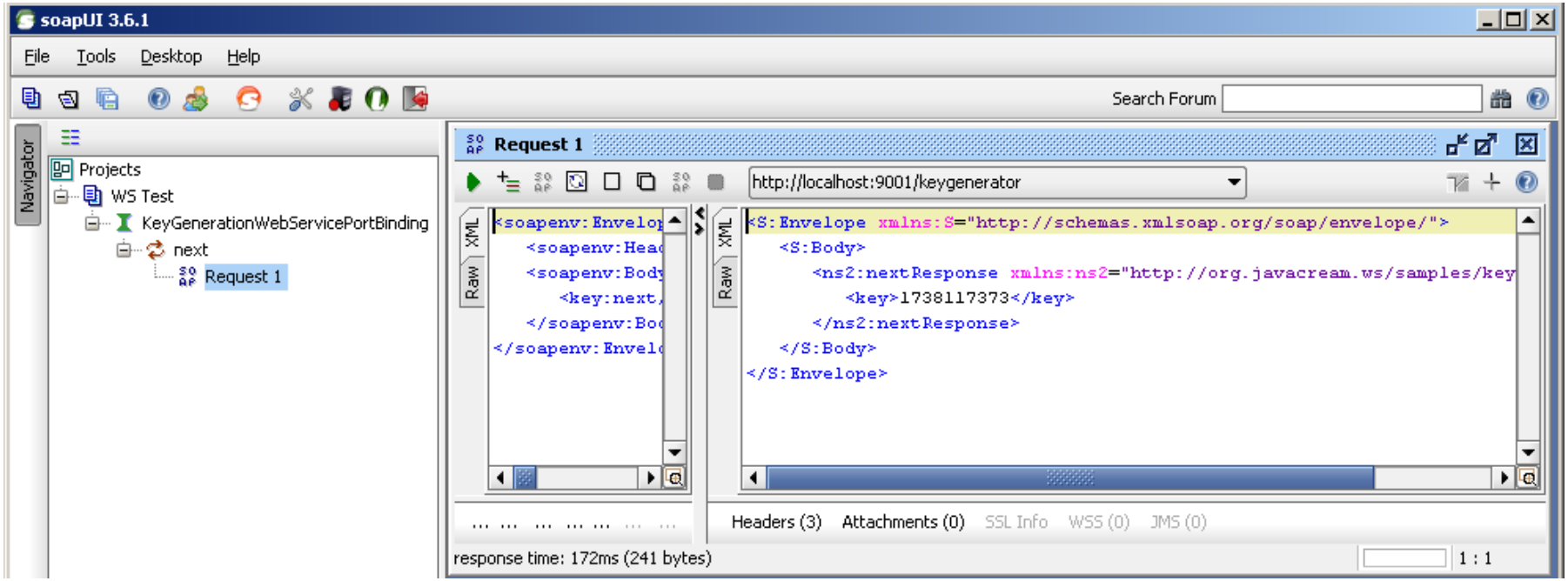
Test mit SOAP UI: SOAP-Request

integrata Cegos Group | inspire qualify change



Test mit SOAP UI: SOAP-Response

integrata Cegos Group | inspire qualify change



5.2

ALTERNATIVE STRATEGIEN ZU CODE FIRST

Probleme mit dem Code-First-Ansatz

integrata Cegos Group | inspire qualify change

- Programmier-Restriktionen müssen eingehalten werden, um Web Services-Einschränkungen zu berücksichtigen
  - Ein Überladen von Methoden ist für SOAP-basierte Web Services nicht vorgesehen.
  - Beschränkung auf WS-kompatible Datentypen
- Echte Business-Logik kann in Daten-Strukturen nicht benutzt werden.
  - Ausnahme hiervon sind Constraint-Validierungen, die von XML-Schema unterstützt werden.
- Bei komplexeren Services mit Benutzerdefinierten Datentypen überdecken die Annotationen die eigentliche Service-Logik
- Eine (unabsichtliche, simple) Änderung in einer Java-Signatur ändert direkt die WSDL
  - Und damit die Service-Schnittstelle!

WSDL wins

- Der Code First-Ansatz kann wie folgt modifiziert werden:
  - Es wird wie bisher eine Klasse mit Web Services-Annotationen erzeugt
  - Wsgen generiert daraus eine WSDL-Datei
    - Diese wird überprüft und bei Bedarf auch noch erweitert und
    - in die Versionsverwaltung mit aufgenommen
- Die WSDL-Generierung auf Server-Seite wird deaktiviert
  - `@WebService(wsdlLocation="URL", …)`
- Inkompatible Änderungen an der Server-Implementierung führen nun zu einer Fehler-Meldung zur Laufzeit
  - `Exception in thread "main" java.lang.Error: Undefined operation name nextKey`

Contract First

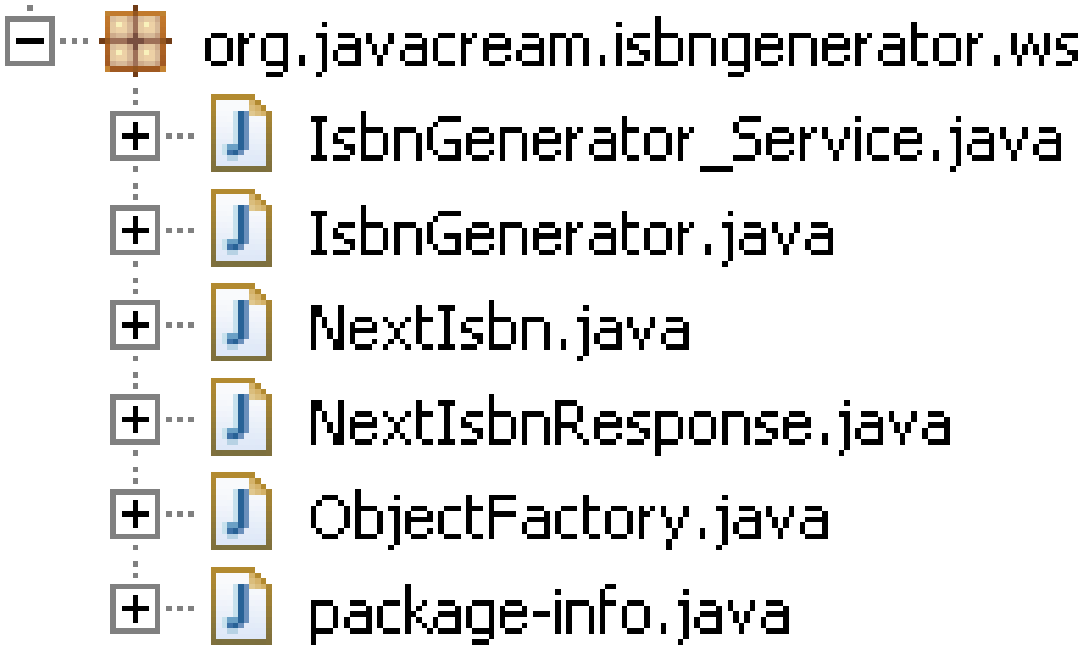
integrata Cegos Group
inspire qualify change

- Im Gegensatz zum Code First-Ansatz wird hier die WSDL als erstes definiert
  - Damit sind Probleme mit nicht-WS-kompatiblen Konstrukten per se ausgeschlossen.
- Der Ablauf bei Contract First wird ist wie folgt:
  - WSDL und XML-Schemata werden modelliert
    - Hierbei können selbstverständlich vorhandene Services und Daten-Strukturen wiederverwendet werden
  - WSDL programmieren
    - Constraints werden mit XML-Mitteln sehr einfach definiert

Contract First: Codegenerierung

integrata Cegos Group | inspire qualify change

- Java Hilfsklassen generieren (ws-import)
  - Es wird eine der WSDL entsprechende Java-Schnittstelle sowie Client-seitige Hilfsklassen erzeugt



Contract First: Server

integrata Cegos Group
inspire qualify change

```
@WebService(name = "IsbnGenerator", targetNamespace =
"http://www.example.org/IsbnGenerator/", wsdlLocation
= "IsbnGenerator.wsdl", endpointInterface =
"org.javacream.isbngenerator.ws.IsbnGenerator",
serviceName = "IsbnGenerator", portName =
"IsbnGenerator")
public class IsbnGeneratorImpl implements
IsbnGenerator {
…
}
```

Contract First: Client

integrata Cegos Group | inspire qualify change

- Der Client benutzt die generierte Schnittstelle und die Hilfsklassen
  - Wie gehabt…

```
IsbnGenerator_Service service = new
  IsbnGenerator_Service();
IsbnGenerator isbnGenerator =
  service.getIsbnGenerator();
```

5.3

FEHLERBEHANDLUNG

Fehlerbehandlung

integrata Cegos Group | inspire qualify change

- Web Services definieren in der WSDL-Spezifikation die Möglichkeit, fault-Datenstrukturen als Rückgabewert einer Operation zu definieren

```
<portType name="BooksWebService">
    <operation name="findBookByIsbn">
      <input message="tns:findBookByIsbn"/>
      <output message="tns:findBookByIsbnResponse"/>
      <fault message="tns:BookWebException"
  name="BookWebException"/>
    </operation>
…
```

Fehlerbehandlung

integrata Cegos Group
inspire qualify change

- Die Umsetzung nach Java macht jedoch ein paar Schwierigkeiten:
  - Exception-Typen sind hier ja relativ komplex, beinhalten gegebenenfalls andere „Caused by“-Exceptions sowie den Stack-Trace.
  - Es ist deshalb notwendig, für Java-basierte Web Services eigene, flache Exception-Typen zu benutzen.
    - Diese treten als simple Daten-Container auf und werden mit @WebFault annotiert

5.4

CLIENT-ZUGRIFF

Generierter Client

- `wsimport` erzeugt einen Satz von Klassen
  - Ein Interface, das der WSDL entspricht
  - Den Datenklassen, die den WSDL-Typen entspricht
  - Eine Hilfsklasse, die als Factory eine Stub-Implementierung der generierten Schnittstelle erzeugt.
  - Die Kommunikationsklassen
- Die Verwendung dieser Hilfsklassen ist einfach:
  - Factory-Klasse erzeugen
  - Aufruf der getter-Methode der zu benutzenden Schnittstelle
  - Benutzung der Schnittstelle

Dynamische Clients

integrata Cegos Group | inspire qualify change

- Statt der Code-Generierung kann der Entwickler selbst die XML „parsen“ und eine kompatible Schnittstelle programmieren
  - Dieses Verfahren ist insbesondere dann sehr einfach, wenn der Client-Programmierer die Quellcodes der Server-Implementierung verfügbar hat
    - Die Annotationen sind dieselben.
- Dann kann mit Hilfe des JaxWs-APIs direkt eine Stub-Implementierung erzeugt werden

```
String urlstr = "http://10.8.2.1:9001/order?wsdl";
URL url = new URL(urlstr);
QName qname = new
  QName("http://org.javacream.ws/samples/order",
  "SimpleOrderService");
Service service = Service.create(url, qname);
OrderWebService orderService = (OrderWebService) service
.getPort(OrderWebService.class);
```

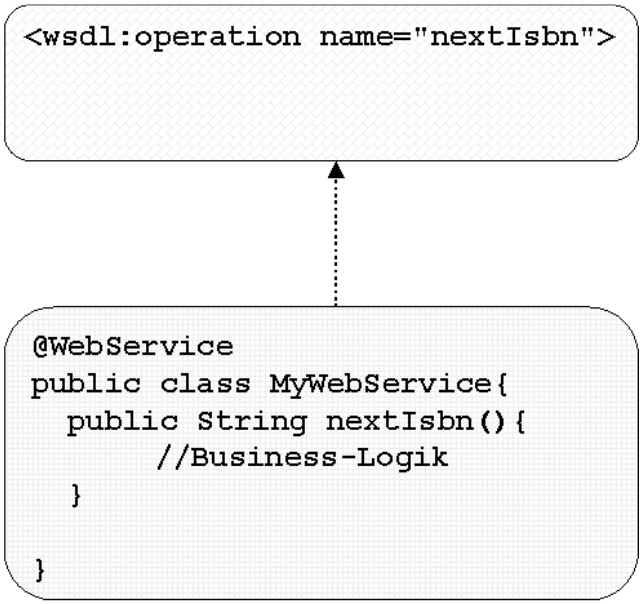
Zugriff mit SAAJ

- SAAJ stellt ein XML-nahes API zur Verfügung, mit dem der SOAP Envelope direkt erzeugt werden kann
- Dieser wird dann über einen simplen http-Request direkt zur Web Service-Implementierung gesendet
  - Ein Beispiel hierzu wird im späteren Kapitel über XML Web Services geliefert

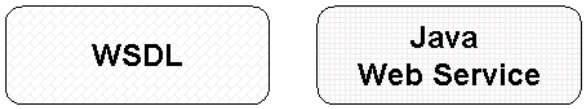
5.5

DESIGN BEISPIELE

Ein pragmatischer Web Service



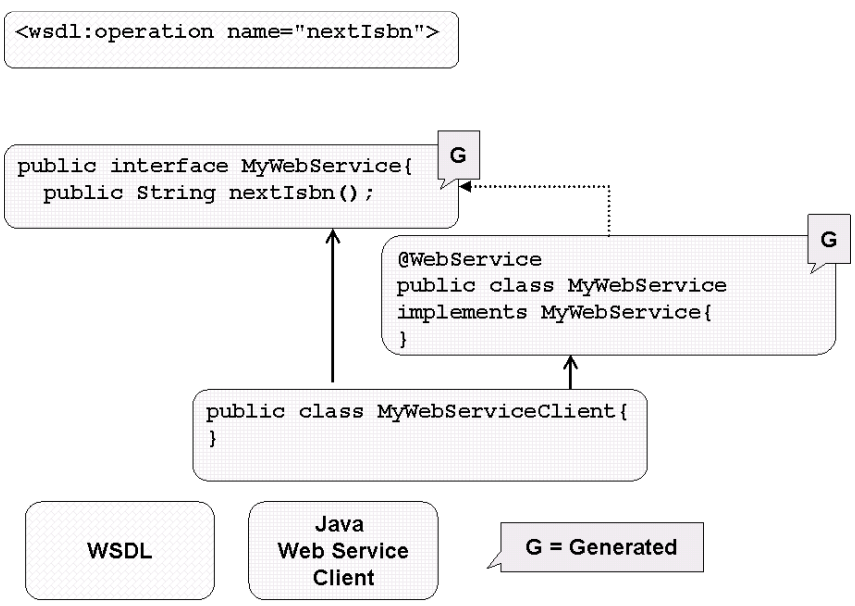
- Hier enthält die Implementierung des Web Services gleichzeitig die Fachlogik
  - Der Implementierungspfeil zwischen der WSDL und der Implementierung ist konzeptionell aufzufassen
- Der Client lässt sich aus der WSDL ein Java-Interface generieren und entwickelt auf dieses hin

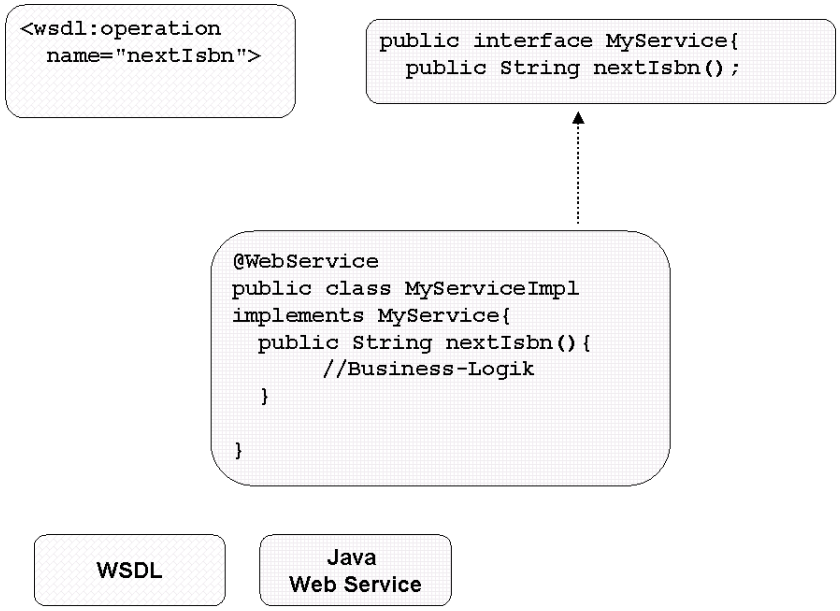


Web Service mit optionaler Schnittstelle

integrata Cegos Group
inspire qualify change

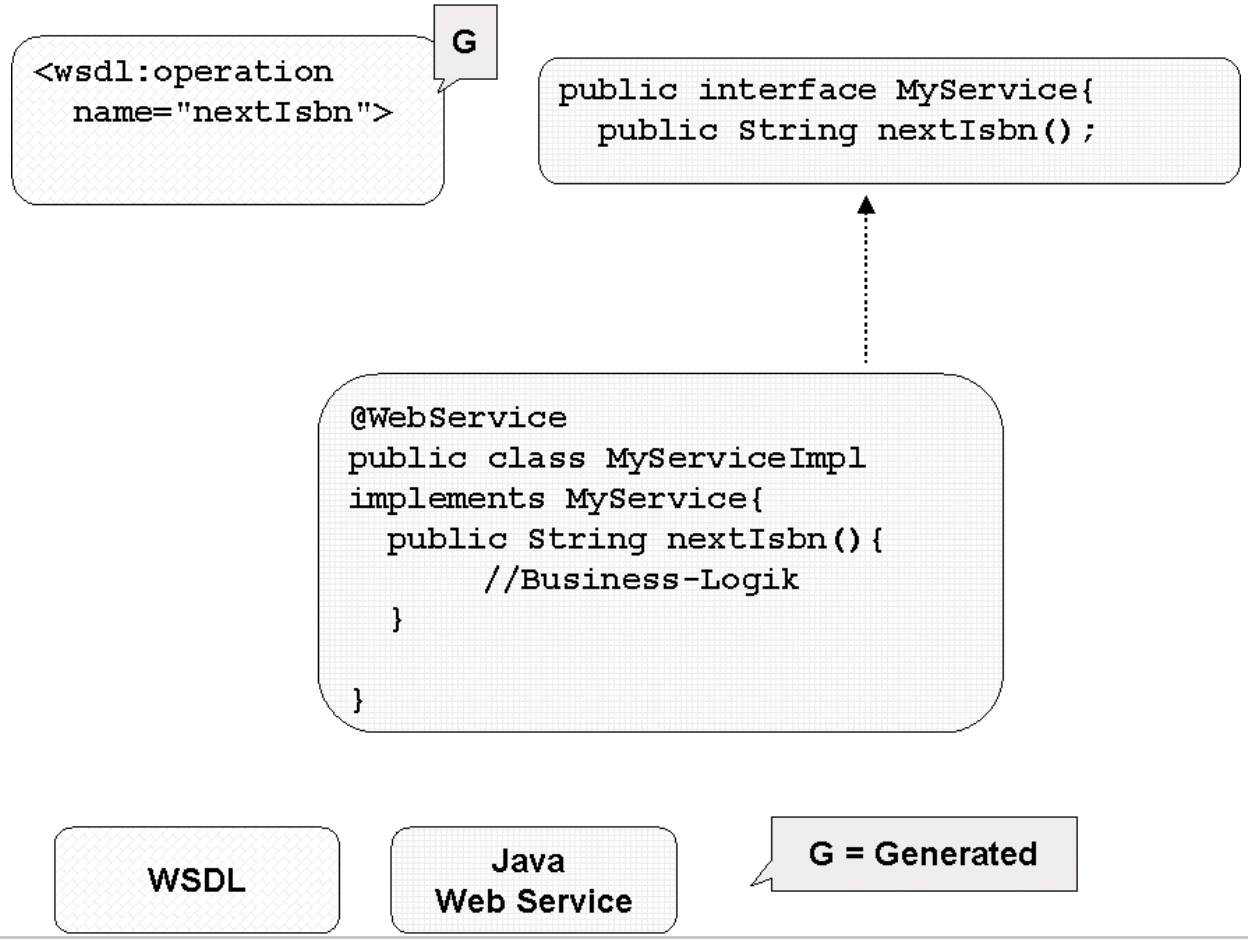
- Natürlich kann der Web Service auch eine Java-Schnittstelle implementieren
- Die Kopplung zwischen Client und Server soll über diese Schnittstelle erfolgen
  - Dies entspricht einem typischen Design für verteilte Anwendungen, wie es beispielsweise auch bei Java RMI vorgeschlagen wird.





Mit Code First

- Hier muss jedoch der Programmierer selbst darauf achten, dass die Java-Schnittstelle und die WSDL korrespondieren
  - Deshalb ist dieser Ansatz häufig bei Code First anzutreffen

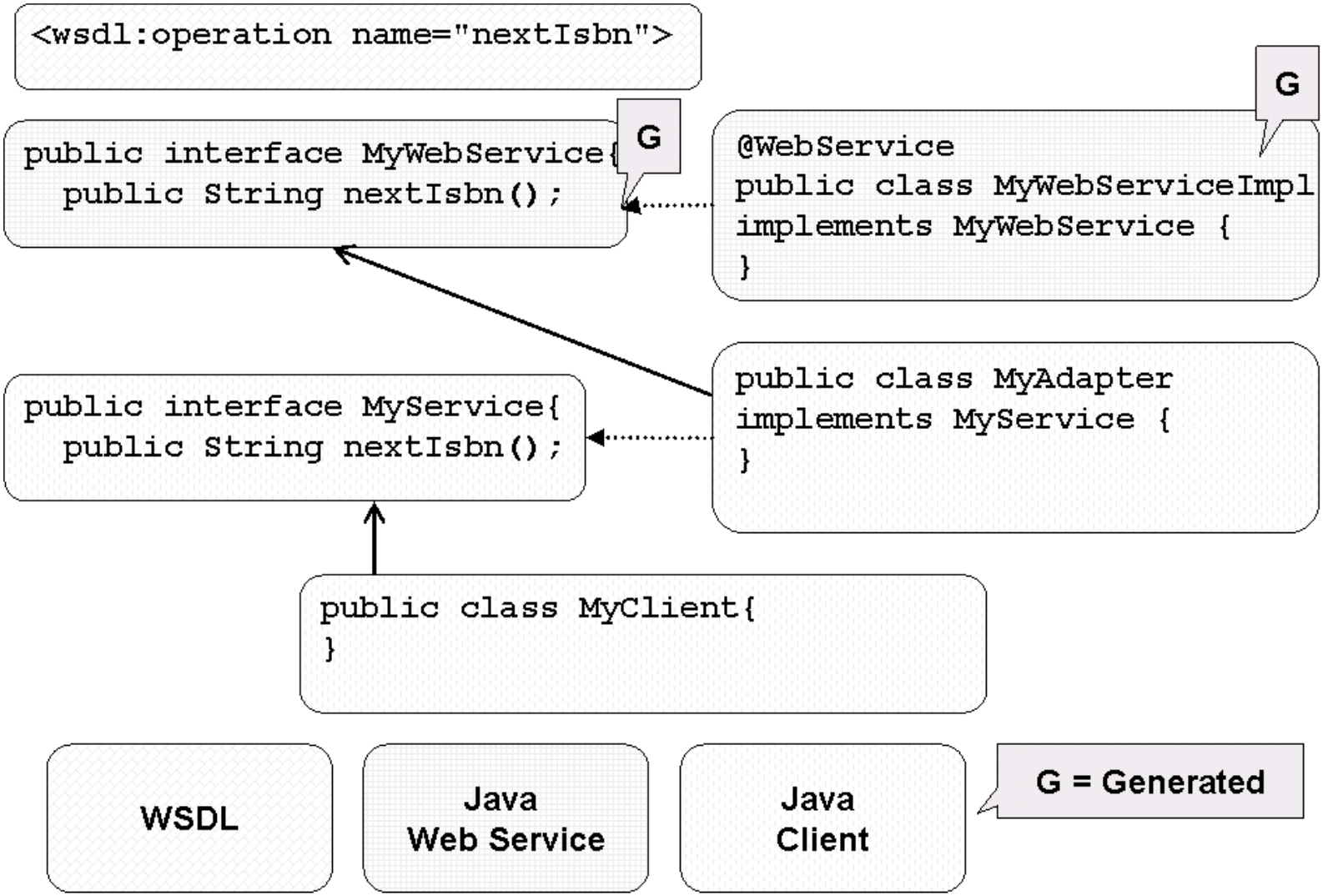


Problem mit Client-Codegenerierung

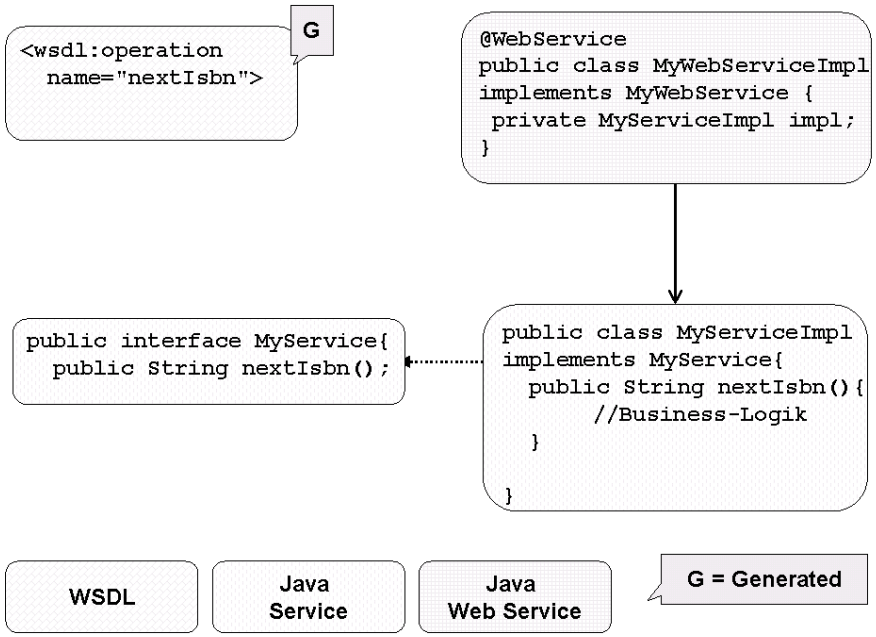
- Die generierten Hilfsklassen implementieren die generierte Schnittstelle
  - Der Client benutzt die generierte Schnittstelle
    - Diese existiert somit quasi doppelt, einmal im Server-Projekt, dann im Client-Projekt
- Der Code-Generator erzeugt gar keine Schnittstelle
  - bzw. diese wird im Rahmen des durchlaufenen Build-Skripts am Ende gelöscht
- Der Client benutzt eine zusätzliche Adapter-Klasse, die zwischen der Server-seitigen Schnittstelle und der generierten Web Service Schnittstelle vermittelt

Adapter

integrata
Cegos Group
inspire
qualify
change



Schichten-Trennung



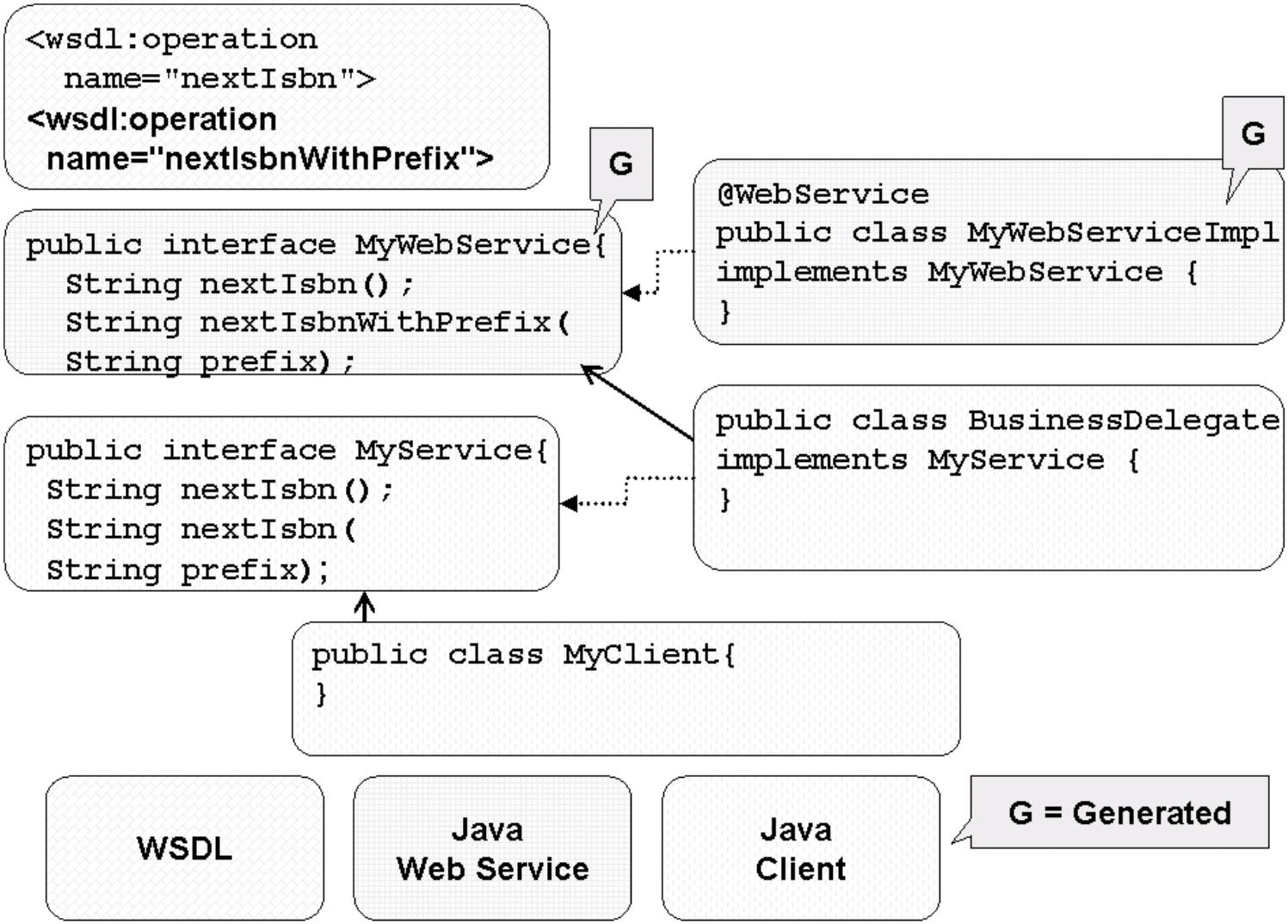
- Es wird unterschieden zwischen den fachlichen Klassen und der Web Services Kommunikationsschicht
  - Die fachliche Implementierung ist vollkommen unabhängig vom JAX-WS-API
  - Die Web Services Implementierung hält eine Referenz auf die Fachklasse und delegiert die Methodenaufrufe weiter

Web Service Facade und Business Delegate

- Die Web Services Implementierung und der Client-Adapter können unabhängig voneinander verändert werden
  - Diese Möglichkeit scheint auf den ersten Blick etwas unsinnig zu sein: Der Web Service soll ja gerade die fachliche Logik aufrufen können
- Was passiert aber, wenn die Fachklasse unabhängig von Web Services modelliert wurde?
  - So könnte beispielsweise die Fachklasse sehr wohl überladene Methoden deklarieren oder aber Klassen des Collection-APIs benutzen
  - Optimierung und Tuning der SOAP-Kommunikation

Web Service Facade und Business Delegate

integrata Cegos Group | inspire qualify change



6

XML-ZENTRIERTE WEB SERVICES

6.1

MOTIVATION

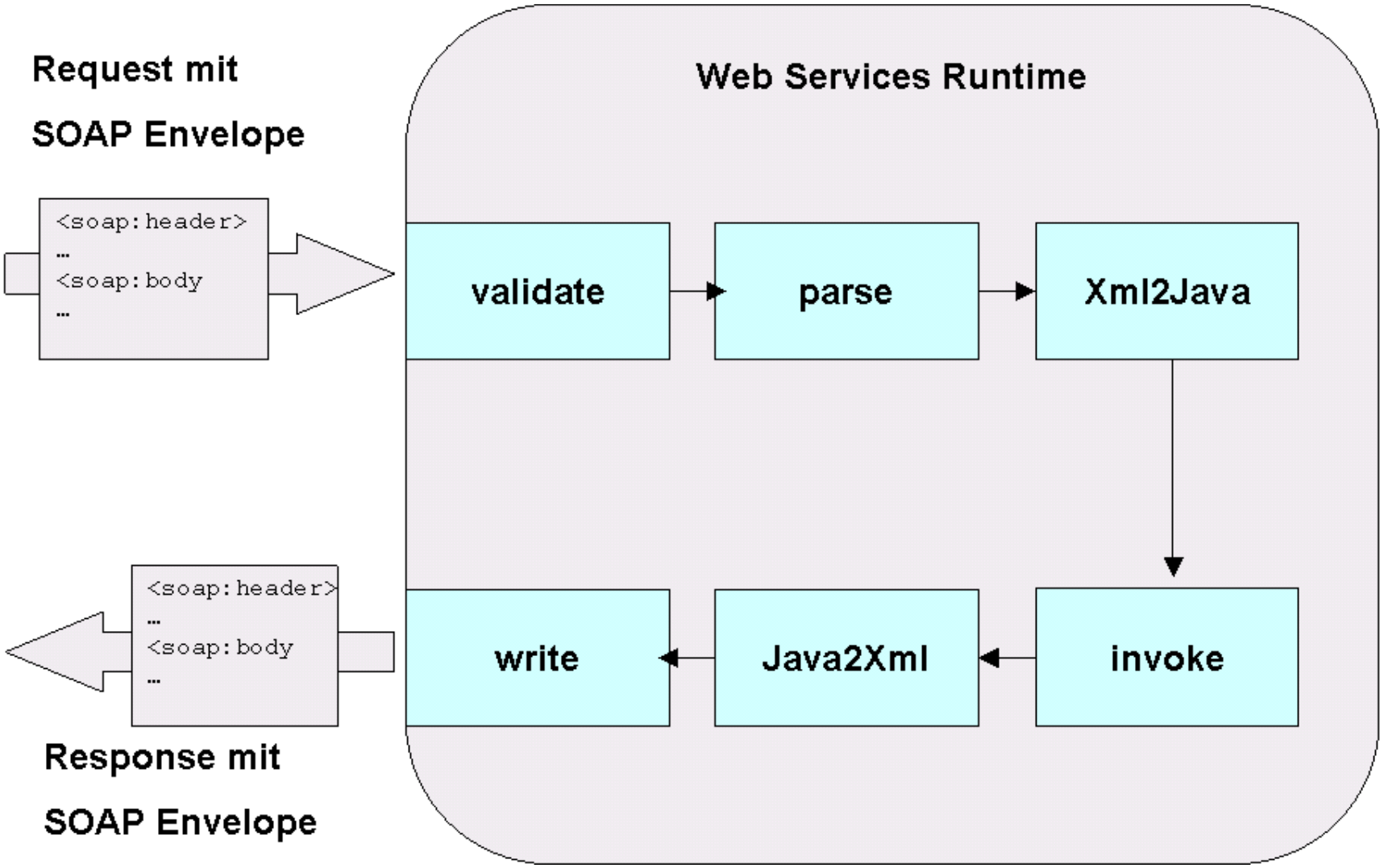
Motivation

- XML eröffnet einige sehr interessante und mächtige Funktionen zur Datenverarbeitung
  - Ein nicht unbeträchtlicher Teil einer typischen Java-Anwendung besteht aus Sequenzen, deren einziges Ziel es ist, Daten aus verschiedenen Quell-Objekten in ein Ziel-Objekt zu kopieren
    - Dies ist mit XML eine einzige simple Transformation
  - Klassen in Java sind statisch mit einem fixierten Daten-Bereich
    - Eine flexible XML-Schema-Definition umgeht dieses Problem komplett
- Zwei Möglichkeiten
  - Das Handler Framework liefert den Durchgriff auf die encodierten Datenstrukturen des SOAP Envelopes in Form von Interceptors
  - Ein `WebServiceProvider` bekommt die übertragenen Nutzdaten als XML-Source

6.2

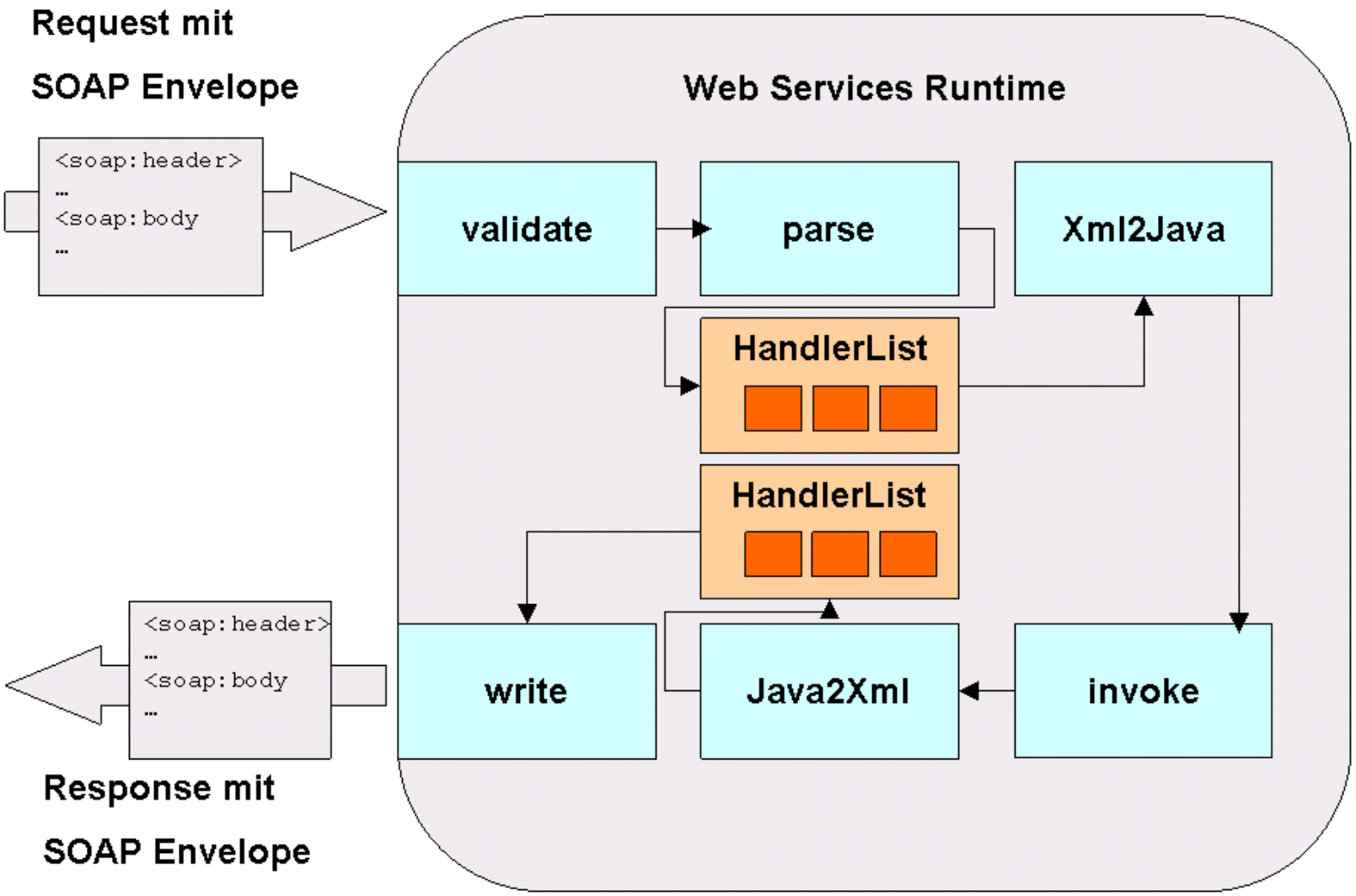
DAS HANDLER FRAMEWORK

Web Services Runtime



Web Services Runtime mit Handlern

integrata Cegos Group
inspire qualify change



Ein einfacher Log-Handler

integrata Cegos Group
inspire qualify change

```
public class LogHandler implements
SOAPHandler<SOAPMessageContext>{
      @Override
      public Set<QName> getHeaders() {}
      @Override
       public void close(MessageContext context) {}
      @Override
       public boolean handleFault(SOAPMessageContext c){}
      @Override
      public boolean handleMessage(SOAPMessageContext c){}
}
```

Konfiguration

integrata Cegos Group | inspire qualify change

- Um einen Handler hinzuzufügen wird eine XML-Konfigurationsdatei erstellt und die Implementierung eingetragen:

```
<handler-chains
  xmlns="http://java.sun.com/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
  ">
<handler-chain>
 <handler>
  <handler-name>JAXWSHandler</handler-name>
  <handler-class>org.javacream.util.ws.LogHandler
  </handler-class>
 </handler>
</handler-chain>
</handler-chains>
```

Aktivierung

integrata Cegos Group
inspire qualify change

- Annotation

```
@WebService(name = "KeyGenerationWebService",
targetNamespace =
"http://org.javacream.ws/samples/keygeneration",
serviceName = "KeyGeneratorService")
@SOAPBinding(parameterStyle =
SOAPBinding.ParameterStyle.WRAPPED)
@HandlerChain(file="handlers.xml")
public class RandomKeyGeneratorWithHandler{
```

- Programmatisch

```
Endpoint.getBinding.setHandlerChain(List<Handler> chain)
Service.setHandlerResolver(HandlerResolver resolver)
```

6.3

WEB SERVICE PROVIDER

Realisierung

integrata Cegos Group | inspire qualify change

- Ausgangspunkt ist eine WSDL
  - Damit wird der Contract First-Ansatz benutzt
- Die Implementierung ist nun ein Web Service Provider
  - Dieser ermöglicht den Zugriff auf den SOAP-Envelope

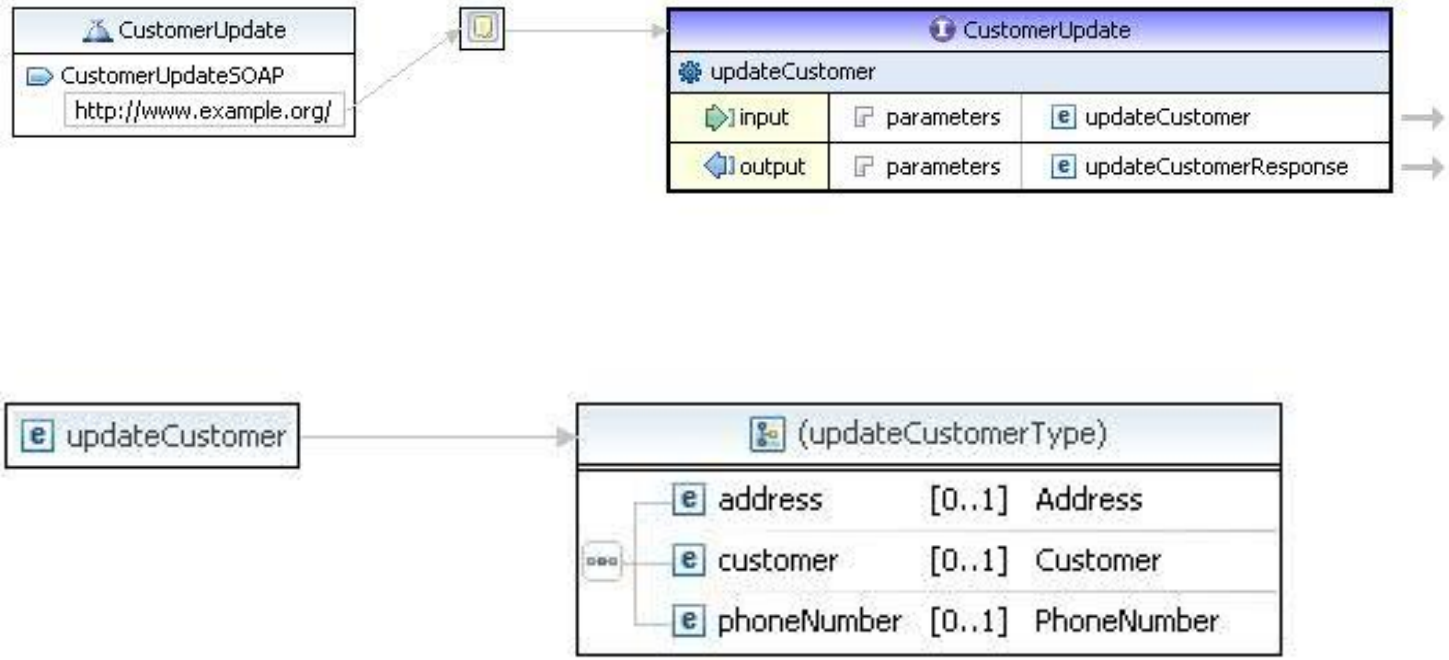
```
@ServiceMode(Mode.PAYLOAD)
@WebServiceProvider(portName="PayloadSOAP",
   targetNamespace="http://www.javacream.org/Payload/",
   wsdlLocation="Payload.wsdl", serviceName="Payload")
public class PayloadProvider implements Provider<Source>{
   private Transformer transformer;
   {
       try {
                transformer =
   TransformerFactory.newInstance().newTransformer();
       } catch (TransformerConfigurationException e) {
                e.printStackTrace();
                throw new IllegalStateException(e.getMessage());
       } catch (TransformerFactoryConfigurationError e) {
                e.printStackTrace();
```

Zugriff mit SAAJ

- Ein Java-Client benutzt das SAAJ-API und erstellt und versendet den SOAP Envelope direkt durch XML-nahe Programmierung
  - Eine Code-Generierung wird für Web Service Provider nicht zufriedenstellend funktionieren!

Beispiel: WSDL und Schema

integrata
Cegos Group
inspire
qualify
change



Beispiel: Der Provider

- Der Provider delegiert einfach an verschiedene Fachklassen weiter, die allesamt ein Document als Parameter erwarten
  - Diese holen sich daraus die jeweils für ihre Aufgaben benötigten Informationen
- Der Provider selbst ist fast trivial:

```
@ServiceMode(Mode.PAYLOAD)
@WebServiceProvider(portName="CustomerUpdateSOAP",
   targetNamespace="http://www.example.org/CustomerUpdate/",
   wsdlLocation="CustomerUpdate.wsdl", serviceName="CustomerUpdate")
public class CustomerUpdateProvider implements Provider<Source>{
   @Override
   public Source invoke(Source source) {
       Document payload = parseInputSource(source);
       addressUpdate.update(payload);
       customerUpdate.update(payload);
       phoneNumberUpdate.update(payload);
       return createResponse(source);
   }
//…
}
```

7

ERGÄNZENDE THEMEN

7.1

ASYNCHRONE CLIENTS

Client-Generierung

integrata Cegos Group | inspire qualify change

- Web Services können auch asynchron aufgerufen werden
  - JAX-WS benutzt hierfür die Multithreading-Möglichkeiten des Clients
- Erweiterung in der WSDL:

```
<bindings
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
    wsdlLocaption="http://localhost:8080/jaxws-
  async/addnumbers?WSDL"
    xmlns="http://java.sun.com/xml/ns/jaxws">
    <bindings node="wsdl:definitions">
        <package name="org.javacream.store.client"/>
        <enableAsyncMapping>true</enableAsyncMapping>
    </bindings>
</bindings>
```

7.2

APPLIKATIONSSERVER

Tomcat

- Tomcat bietet selber keine native Unterstützung für Web Services an
  - Um einen Web Service zu installieren muss deshalb eine geeignete Servlet-Implementierung installiert werden
  - Alternativ: Verwendung von TomEE
- Die Web Services werden aus der `sun-jaxws.xml` gelesen

- Das Deployment eines Web Services erfolgt durch die Definition einer Stateless Enterprise JavaBean

```
@Stateless
@WebService(name = "KeyGenerationWebService",
targetNamespace =
"http://org.javacream.ws/samples/keygeneration",
serviceName = "KeyGeneratorService")
@SOAPBinding(parameterStyle =
SOAPBinding.ParameterStyle.WRAPPED)
public class RandomKeyGenerator{
…}
```

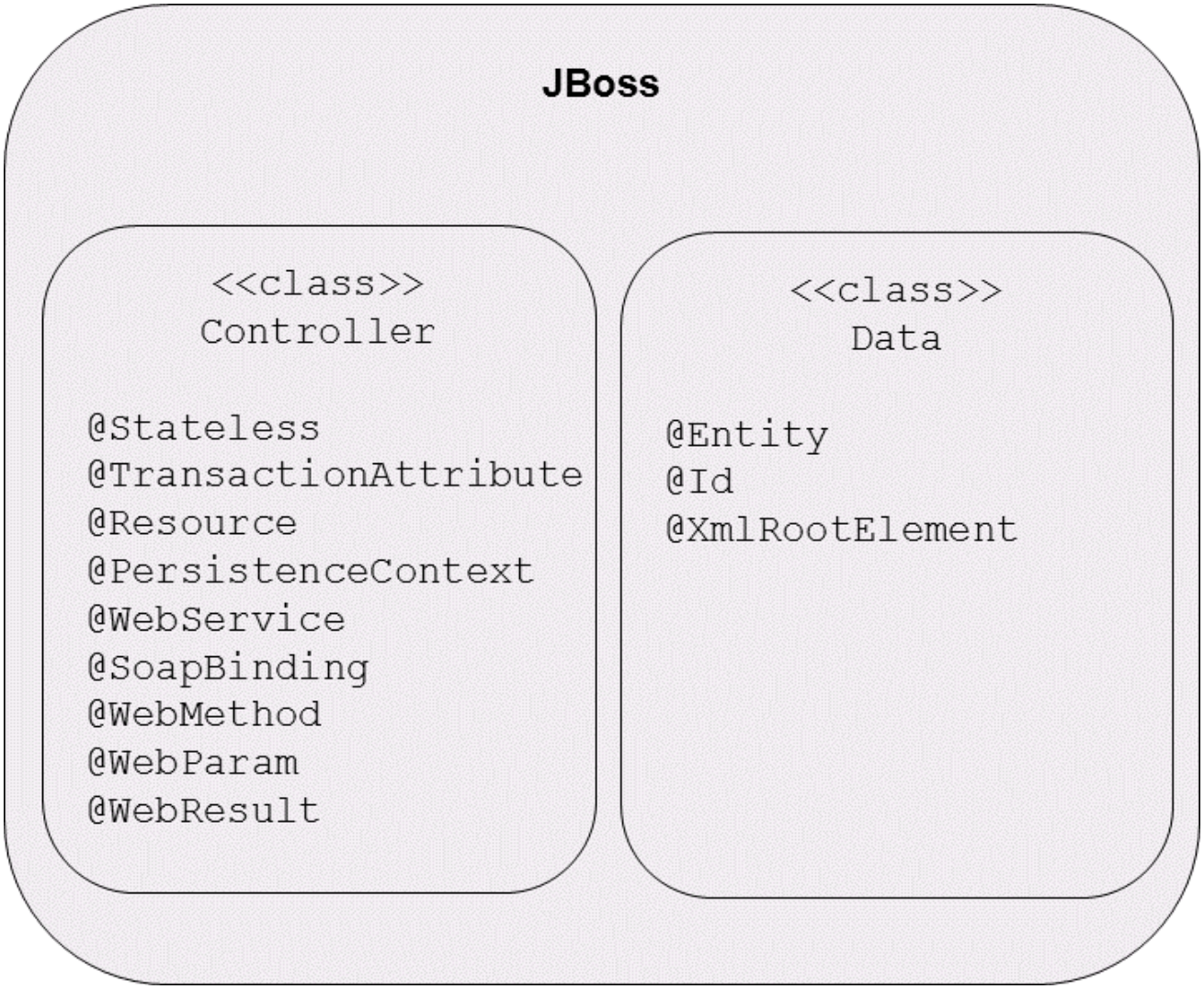
- Damit stehen dem Web Service die Dienste des Applikationsservers, insbesondere das Transaktionsmanagement zur Verfügung

7.3

ARCHITEKTUREN

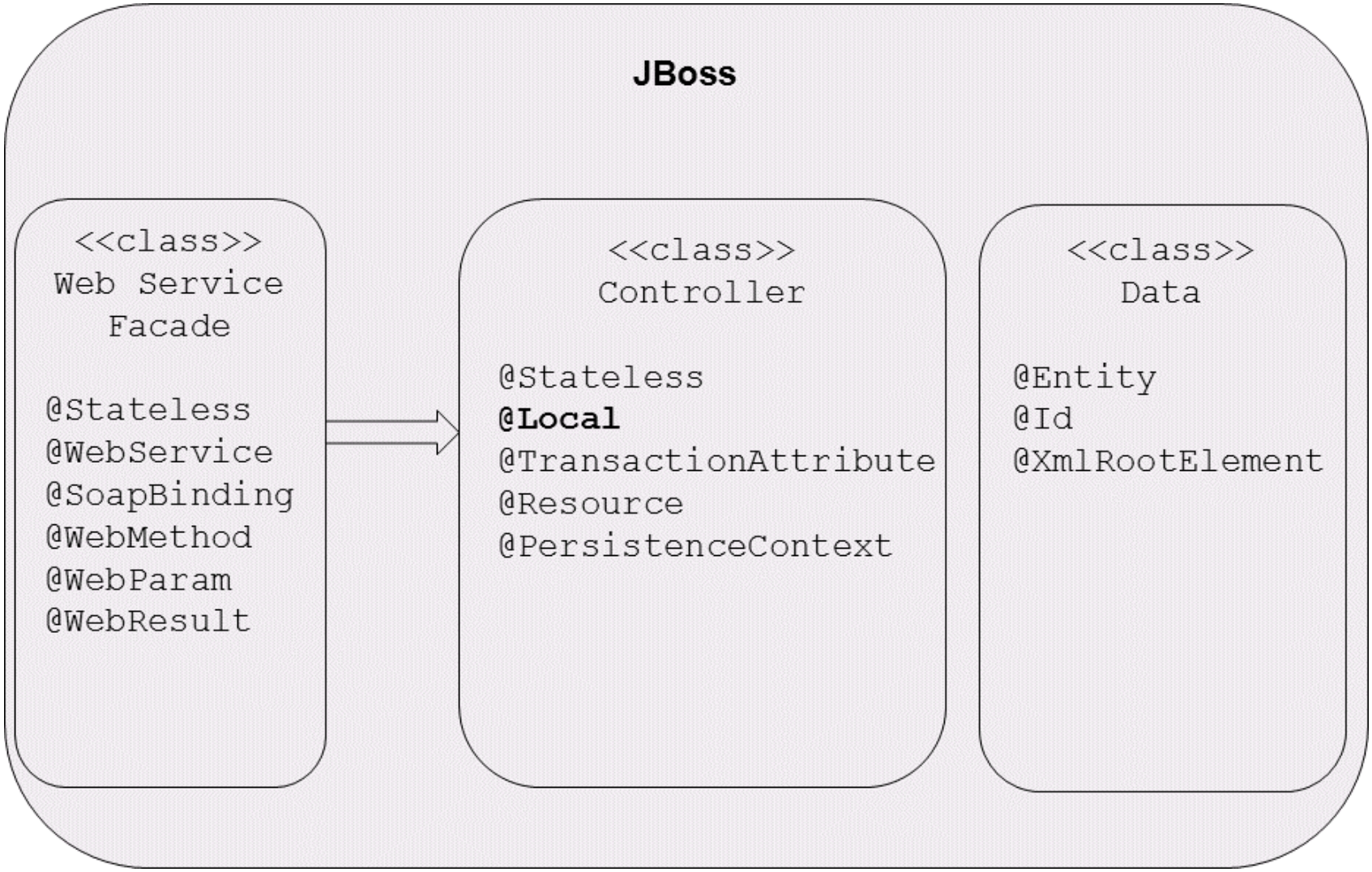
Architekturen: All In One

integrata Cegos Group
inspire qualify change



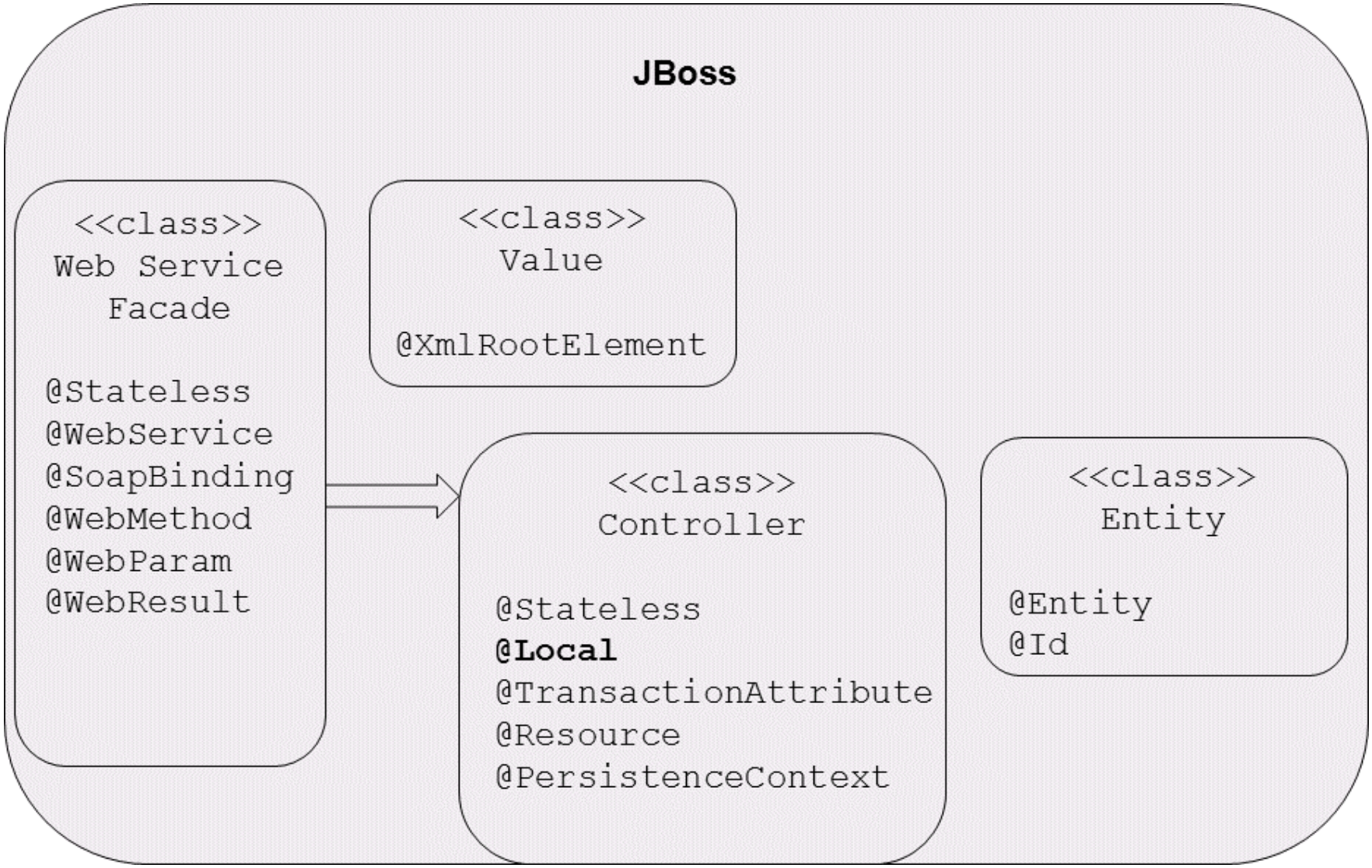
Web Service Facade

integrata Cegos Group
inspire qualify change



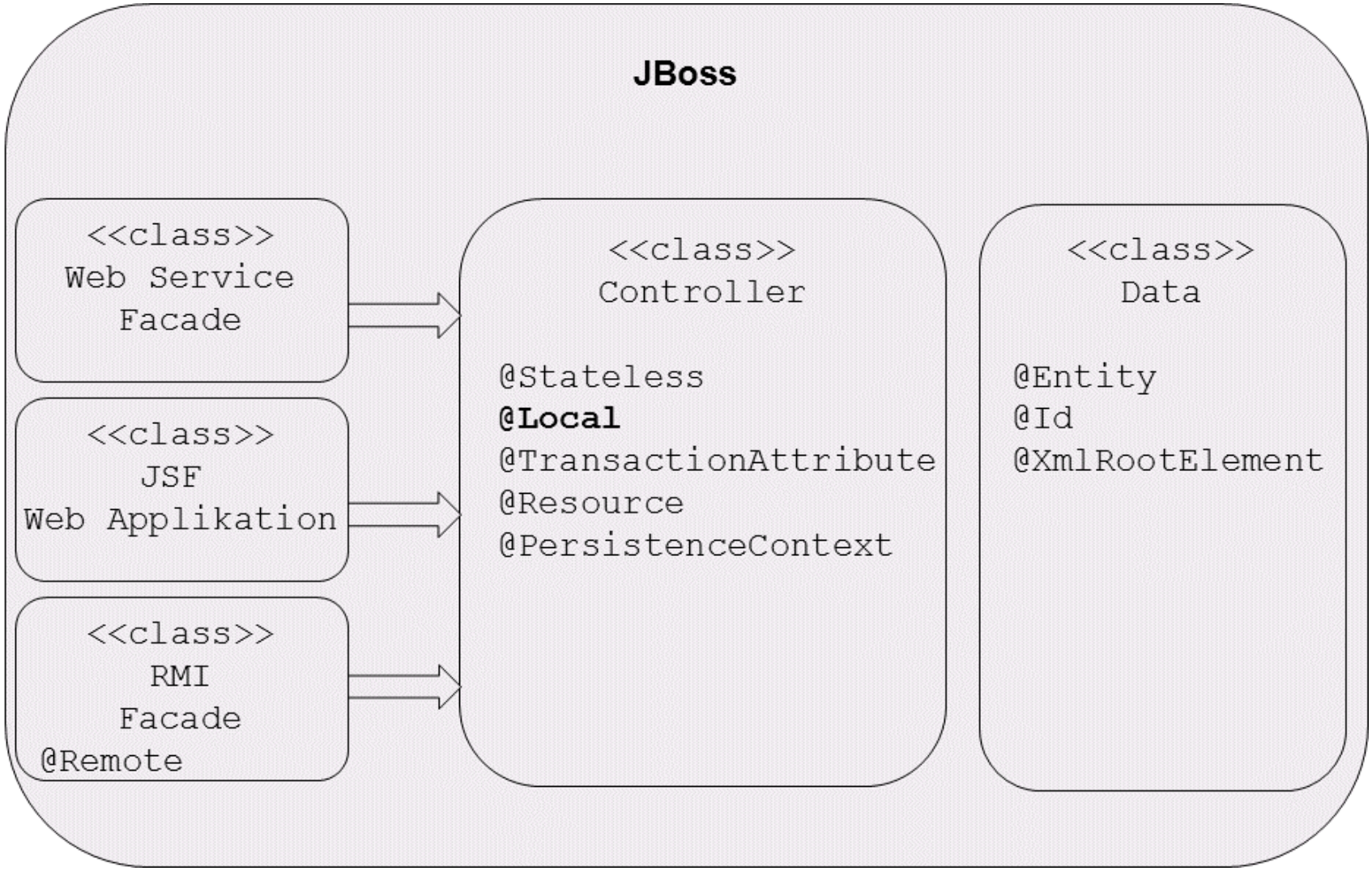
Facade und Web Service Valie

integrata Cegos Group
inspire qualify change

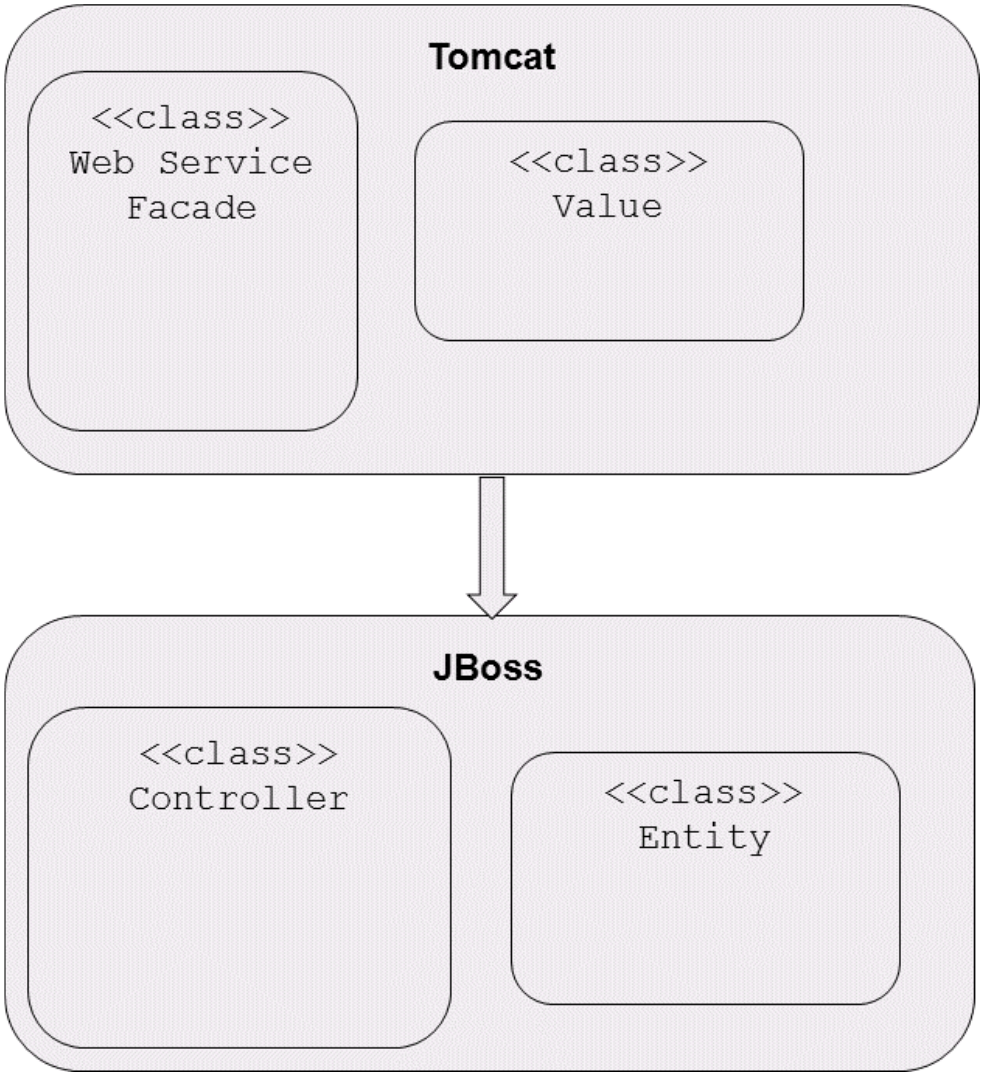


Wiederverwendung des Controllers

integrata Cegos Group | inspire qualify change



Web Service Server

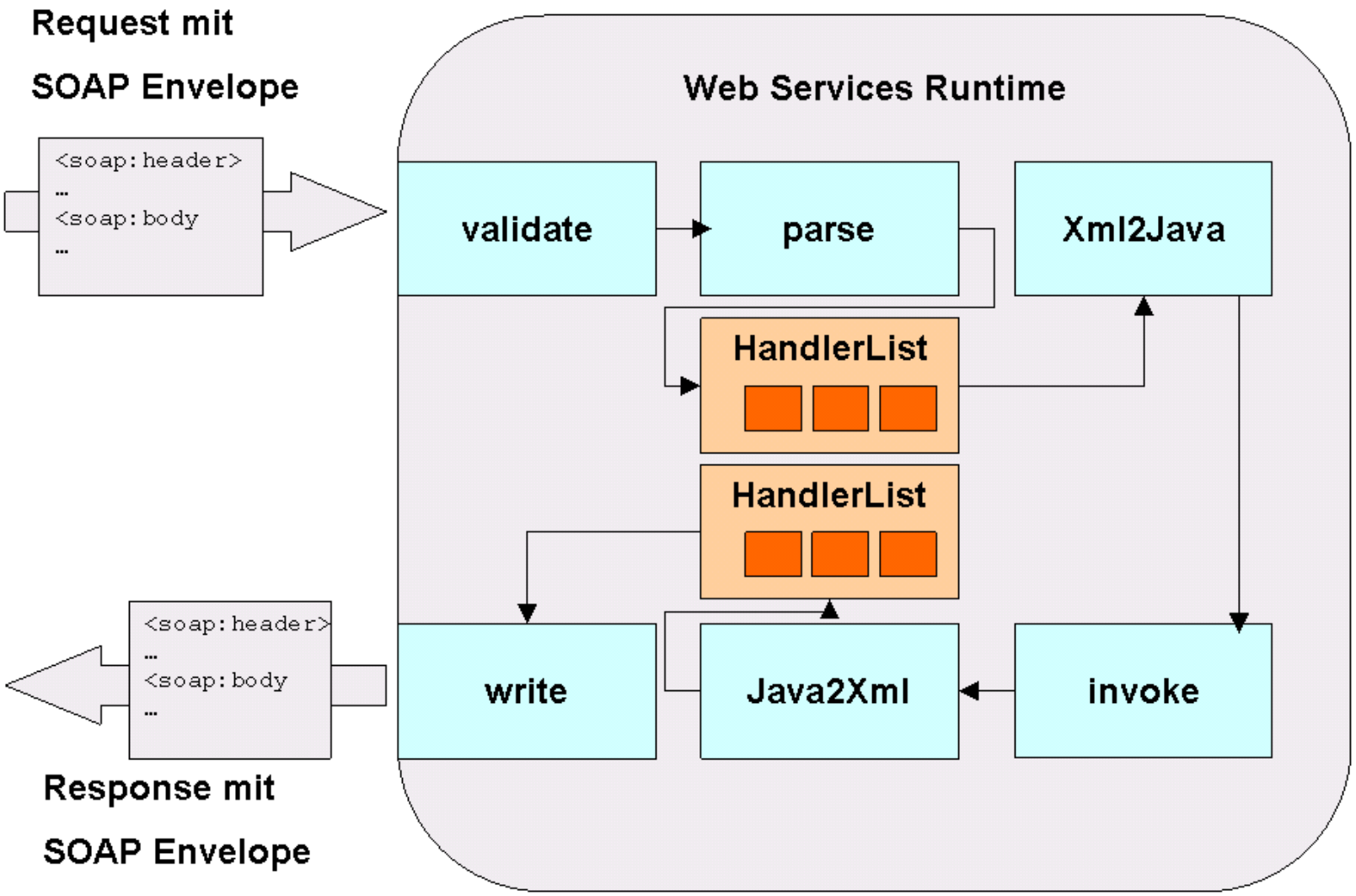


7.4

UMGANG MIT GROßEN DATENMENGEN

Alternative Encodierung der Nutzdaten

- Selbst-definierte oder proprietäre Formate sind selbstverständlich auch möglich
  - So kann beispielsweise ein serialisiertes Java-Objekt übertragen werden
- Verwendung von Referenzen
  - Statt des Objektes selber wird eine Referenz, z.B. auf eine Lokation innerhalb eines File Servers, übertragen
  - Dieses Konzept ist Grundlage des MTOM-Konzeptes, das im nächsten Abschnitt behandelt wird


Request mit
SOAP Envelope
<soap:header>
…
<soap:body
…
Web Services Runtime
validate
parse
Xml2Java
HandlerList
HandlerList
write
Java2Xml
invoke
<soap:header>
…
<soap:body
…
Response mit
SOAP Envelope

MTOM-Feature

- @MTOM(enabled = true, threshold = 1)
- Der `threshold` ist eine Angabe in Byte
  - Überschreitet die Größe der zu übertragenden Nutzdaten diesen Wert, so schaltet der Web Service auf MTOM um
- Client

```
OrderService orderService = new OrderService();
    OrderWebService orderWebService =
orderService.getOrderWebServicePort(new MTOMFeature(1));
```

7.5

SECURITY

Ebenen der Security

- Verschlüsselung der Kommunikation
  - Diese kann bereits von der normalen Java Runtime gewährleistet werden
  - Diese unterstützt SSL-Verbindungen und verwaltet einen KeyStore mit Identitäten und/oder Zertifikaten
- Authentifizierung
  - Dazu benutzen Web Services die Standard-Mechanismen des Applikationsservers
    - Enterprise Java Beans benutzen die `@RolesAllowed`-Annotation.
    - Servlets definieren in der web.xml ein `<security-con-straint>`
  - Das Setzen von Username/Password erfolgt auf Client-Seite durch den `BindingProvider`

Höherwertige Anforderungen

- Verschlüsselung der Nutzdaten unabhängig von SSL
  - Dies ist in Java einfach möglich, da XML Encryption/Decryption integriert ist
- Aufnahme der Security-Informationen in Header-Properties des SOAP Envelopes
  - Diese Erweiterung des Envelope-Schemas ist Bestandteil der WS-Security-Spezifikation.
- WS Policy legt in der WSDL notwendige Authentifizierungs-Routinen fest

Ein Security Proxy

integrata Cegos Group
inspire qualify change