JAX-WS

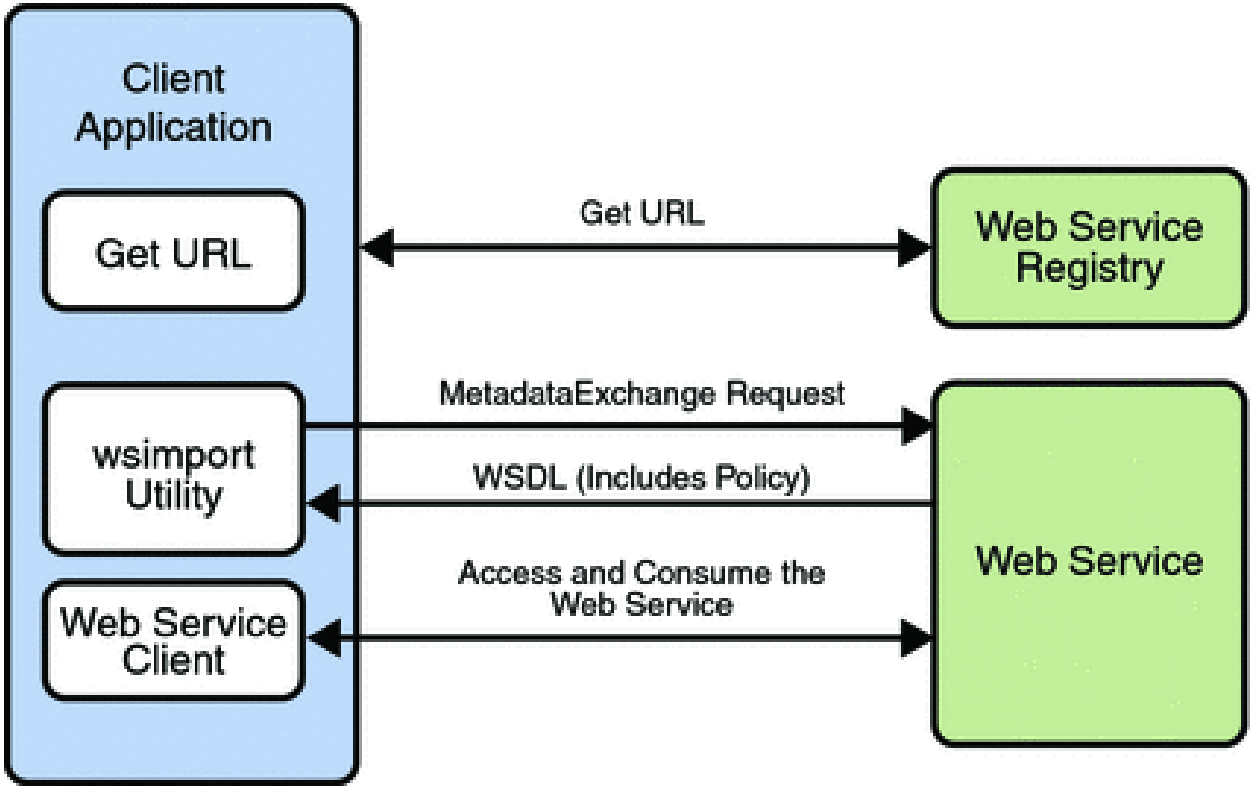
Java API for XML Web Services

SS15 | Prof. Dr. Hendrix | IM14 | 03.06.2015 | Christopher Schulze und Sebastian Nemak

Gliederung

- Überblick
- Spezifikation
- JAX-RPC vs. JAX-WS
- Umsetzung eines Client
- Handler
- JAX-WS in Java
- JAX-WS Runtime
- Umsetzung eines Service
- Annotations und Implementierung
- Live Demo
- Zusammenfassung

JAX-WS - Überblick



http://oppansource.com/wp-content/uploads/2014/07/bootstrappingConfiguration.png

Spezifikation (1)

- Nachfolger von JAX-RPC
- WSDL 1.1
- SOAP 1.1 und 1.2
- JAX-WS 2.0 (JSR 224, April 2006, abwärtskompatibel)

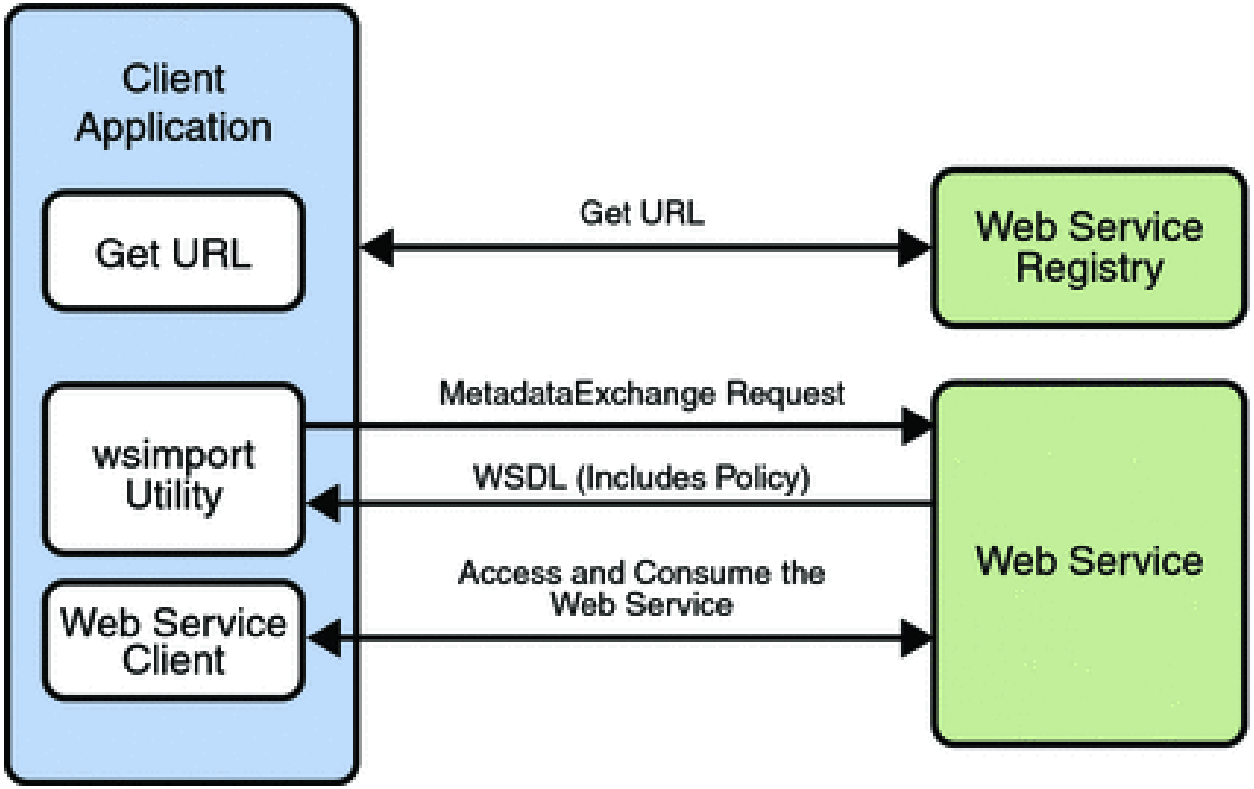
Spezifikation (2)

- Client-/Service APIs
- Core APIs: Bindings und Exceptions
- Annotations
- Handler
- MEP - Message Exchange Pattern
- MTOM - SOAP Message Transmission Optimization Mechanism

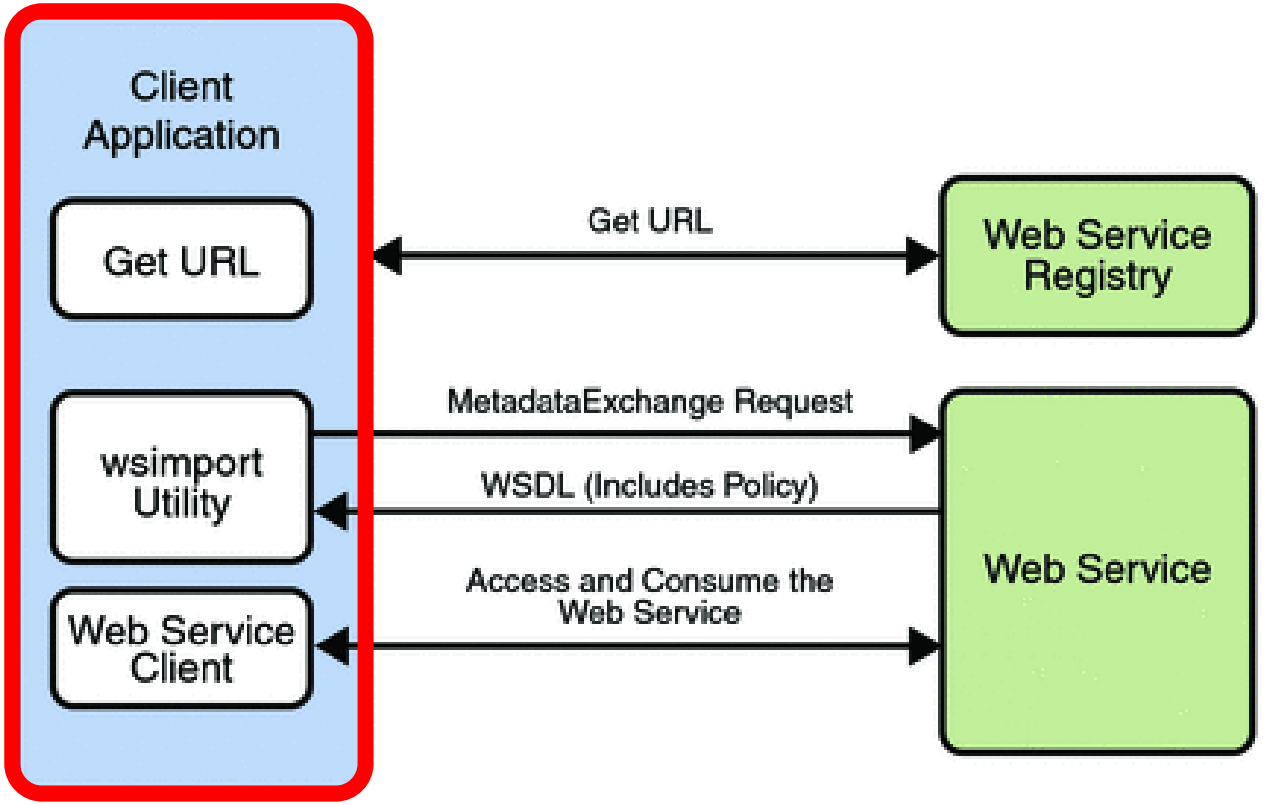
JAX-RPC vs. JAX-WS

	JAX-RPC	JAX-WS 2.0
JDK-Version	ab 1.4	ab 1.5
Annotations	-	vorhanden
SOAP-Version	SOAP 1.1	SOAP 1.1 und 1.2
Schema-Mapping	eigenes	JAXB
REST	-	unterstützt
Generator-Tools	wsdl2java und java2wsdl	wsimport
MEP	nur synchrone Aufrufe	auch asynchrone Aufrufe
MTOM	-	unterstützt

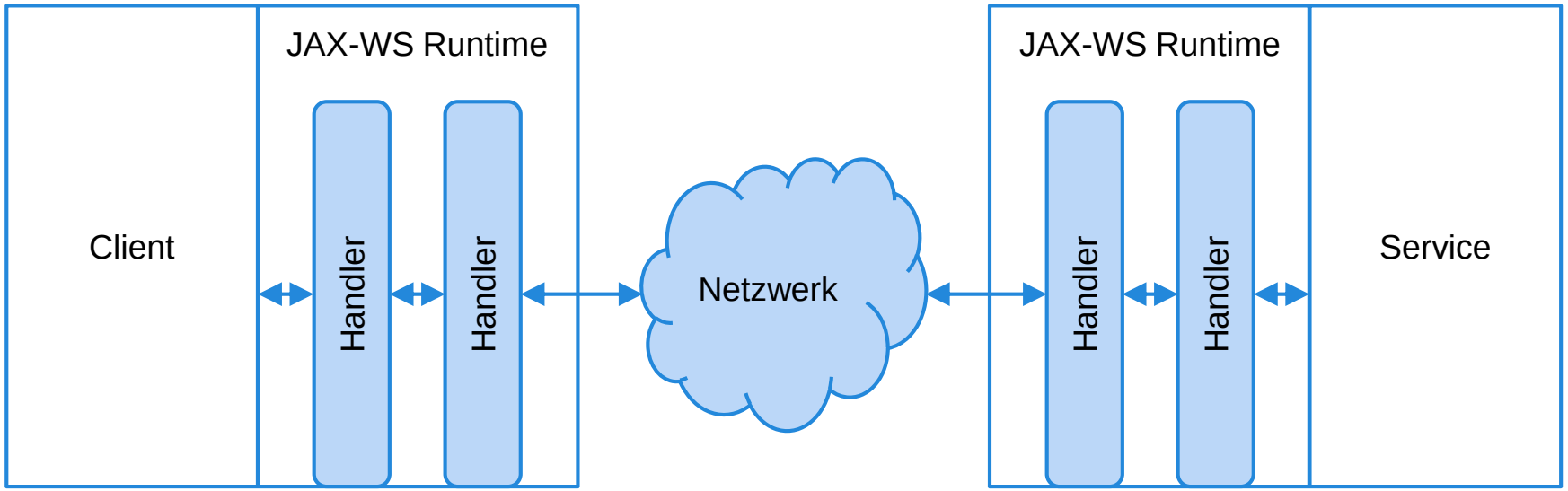
JAX-WS - Client



JAX-WS - Client



Handler


Client
JAX-WS Runtime
Handler
Handler
Netzwerk
JAX-WS Runtime
Handler
Handler
Service

Handler - Zweck

- Übertragung von Metainformationen
- Schutz der Nachricht
- Entwicklungsunterstützung
- Verändern der Nachricht
- Messung der Service-Qualität

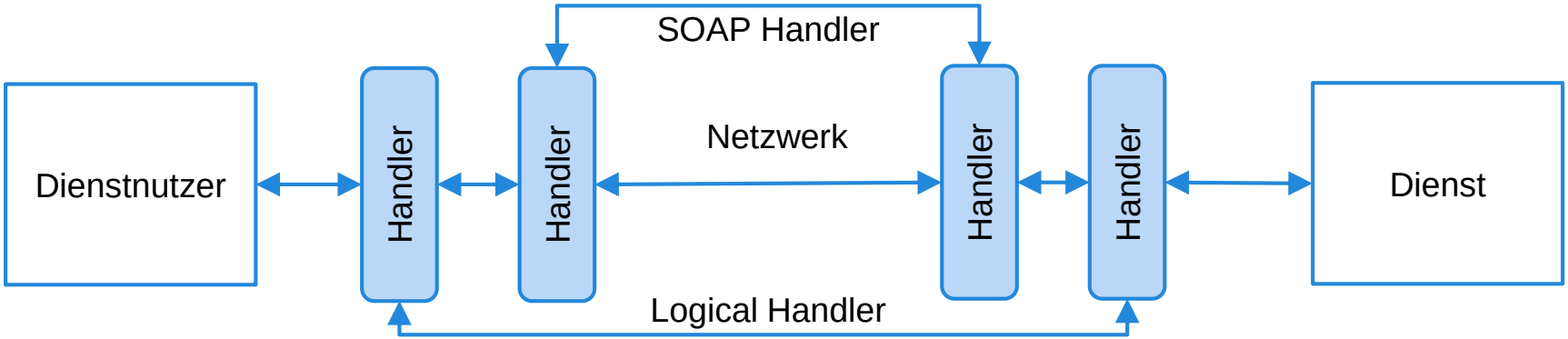
Handler - JAX-WS

- Einheitliche Schnittstelle:

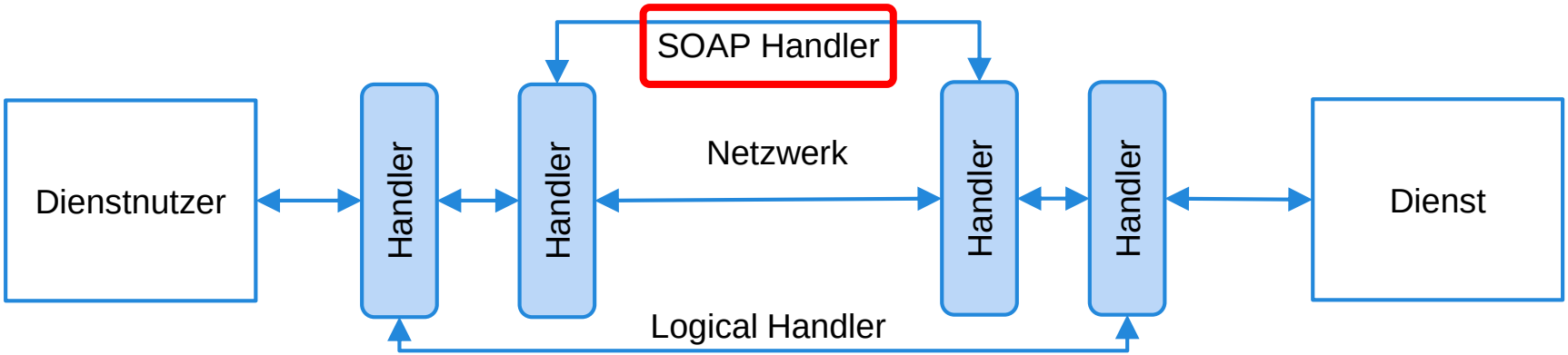
javax.xml.ws.handler.*

javax.xml.ws.handler.soap.SOAPHandler
javax.xml.ws.handler.LogicalHandler

Handler - Arten

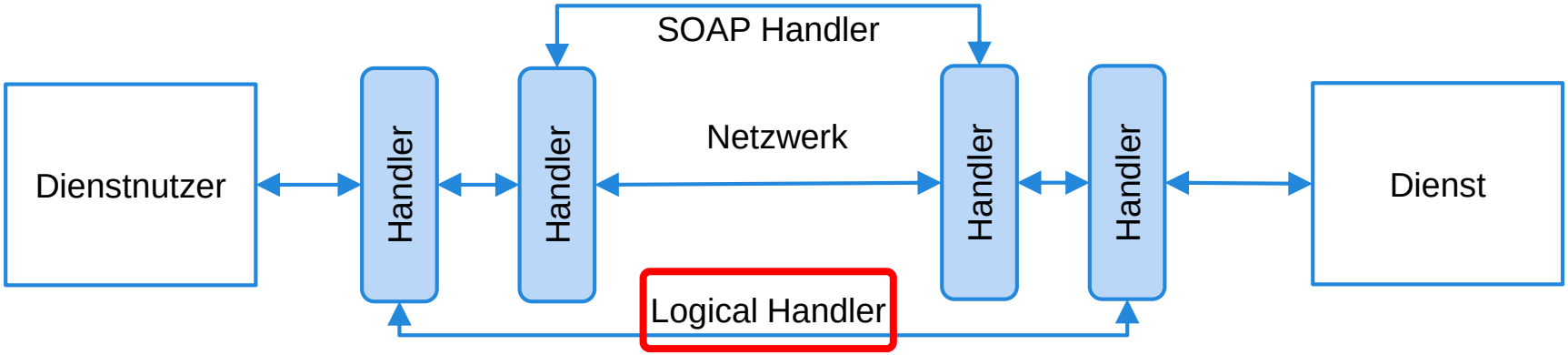


Handler - Arten - SOAP-Handler



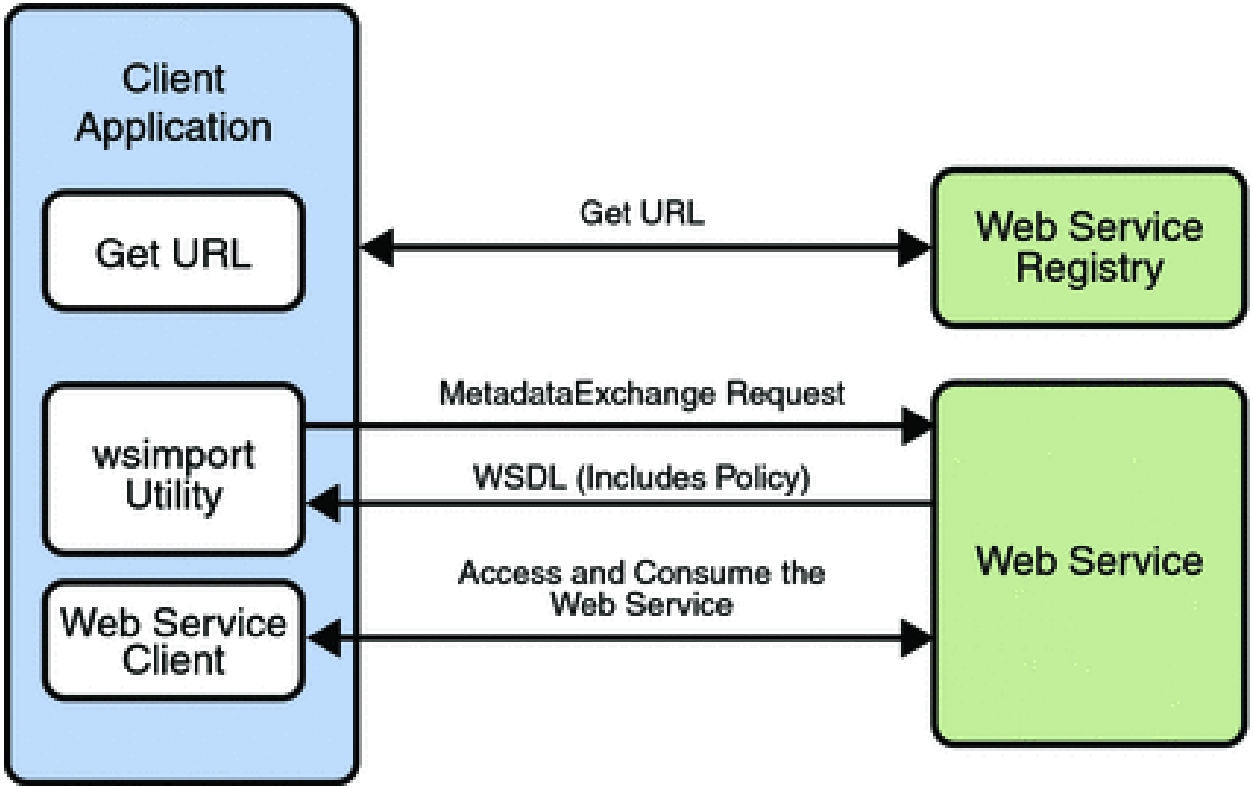
- Einfluss auf gesamte SOAP-Message
- z.B. für Security-Handler

Handler - Arten - Logical-Handler

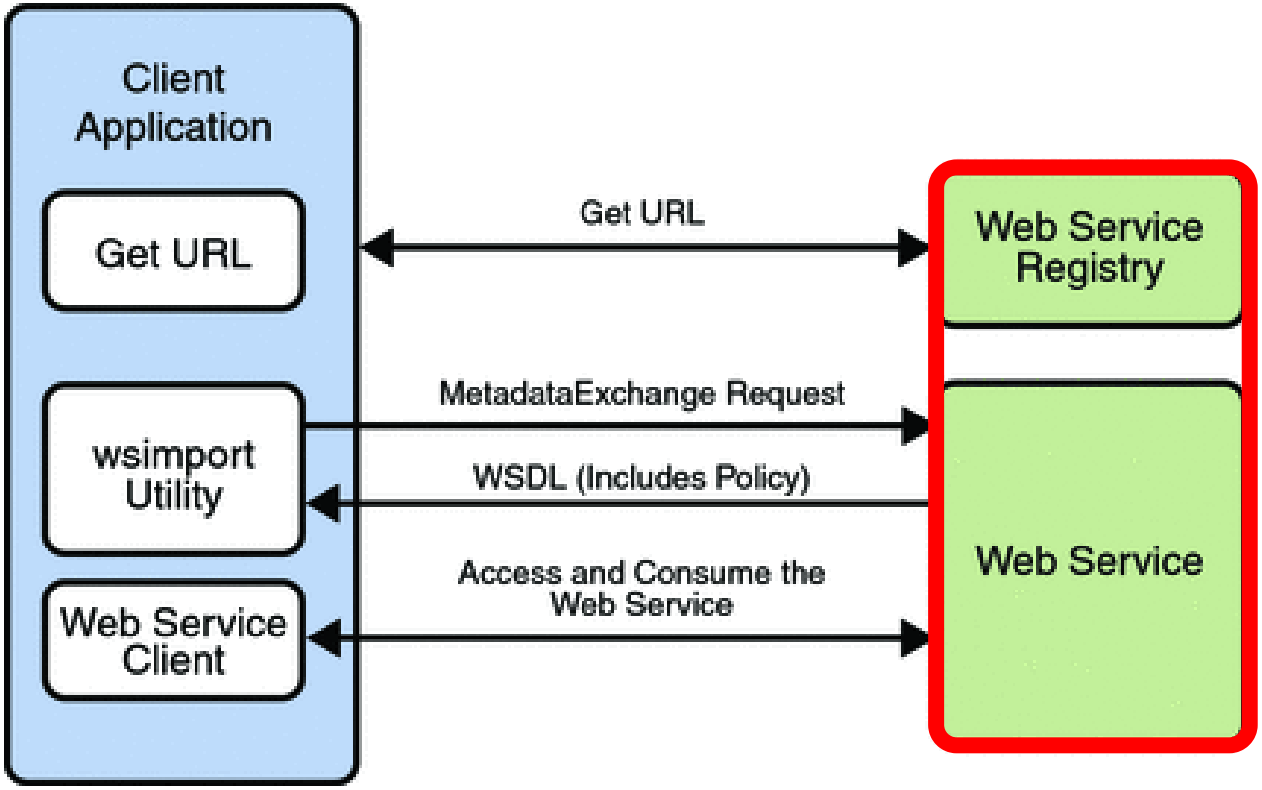


- Einfluss auf Nutzdaten
- z.B. Performance-Monitor, Logging

JAX-WS - Service



JAX-WS - Service



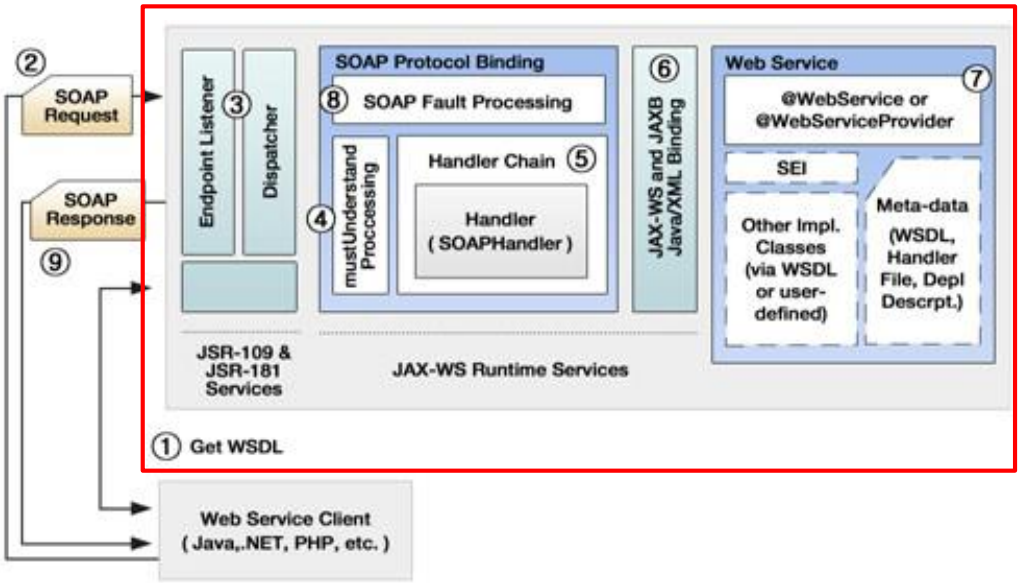
JAX-WS in Java

- Java Standard Edition (SE)
  - Bereitstellung des Service Endpoint Interface (SEI)

```
public static void main(final String[] args) {
        Endpoint endpoint = javax.xml.ws.Endpoint.create(new ExamplePort());
        endpoint.publish("http://localhost:8080/ExampleService");
}
```

- Java Enterprise Edition (EE)
  - Application Server stellt JAX-WS Runtime bereit
    - z.B. GlassFish, JBoss, Wildfly, Weblogic, ...

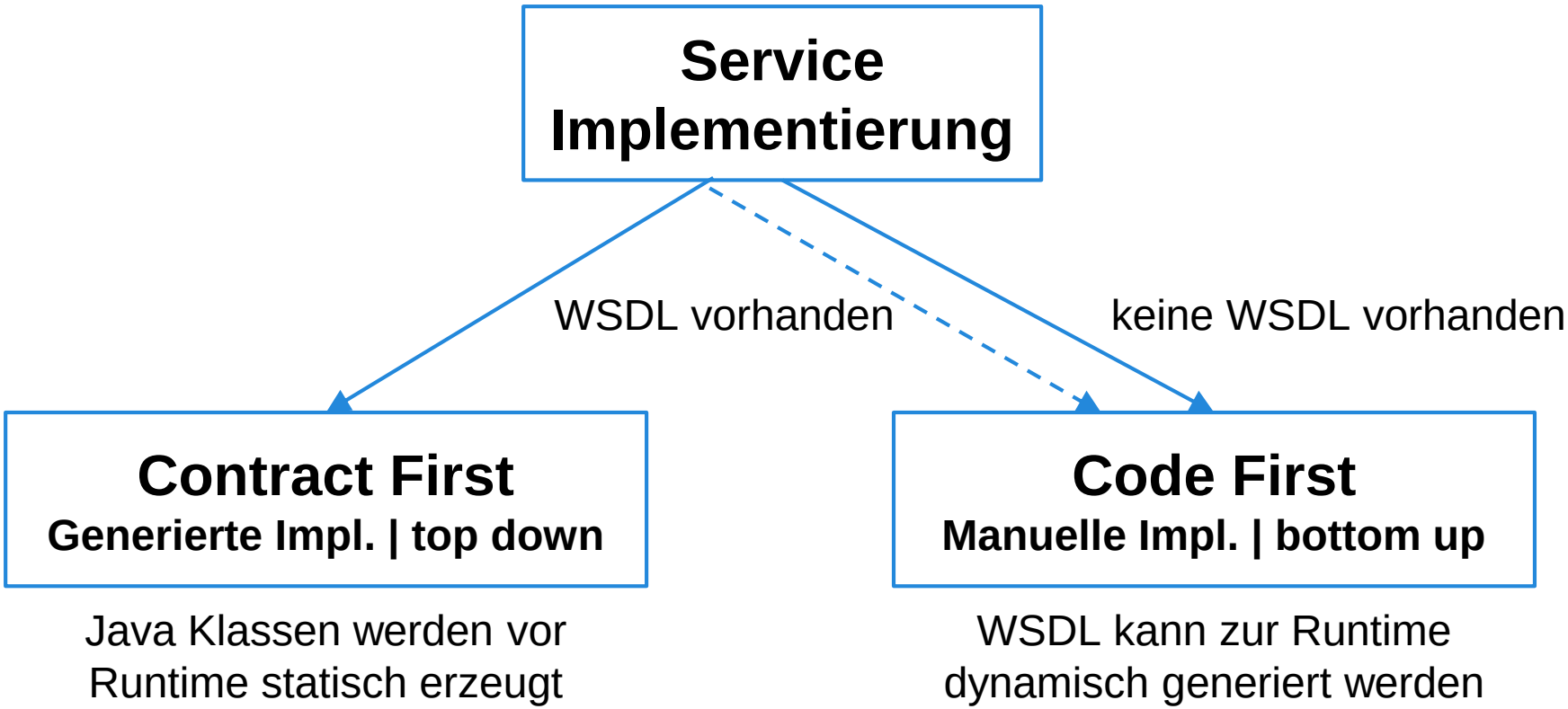
JAX-WS Runtime



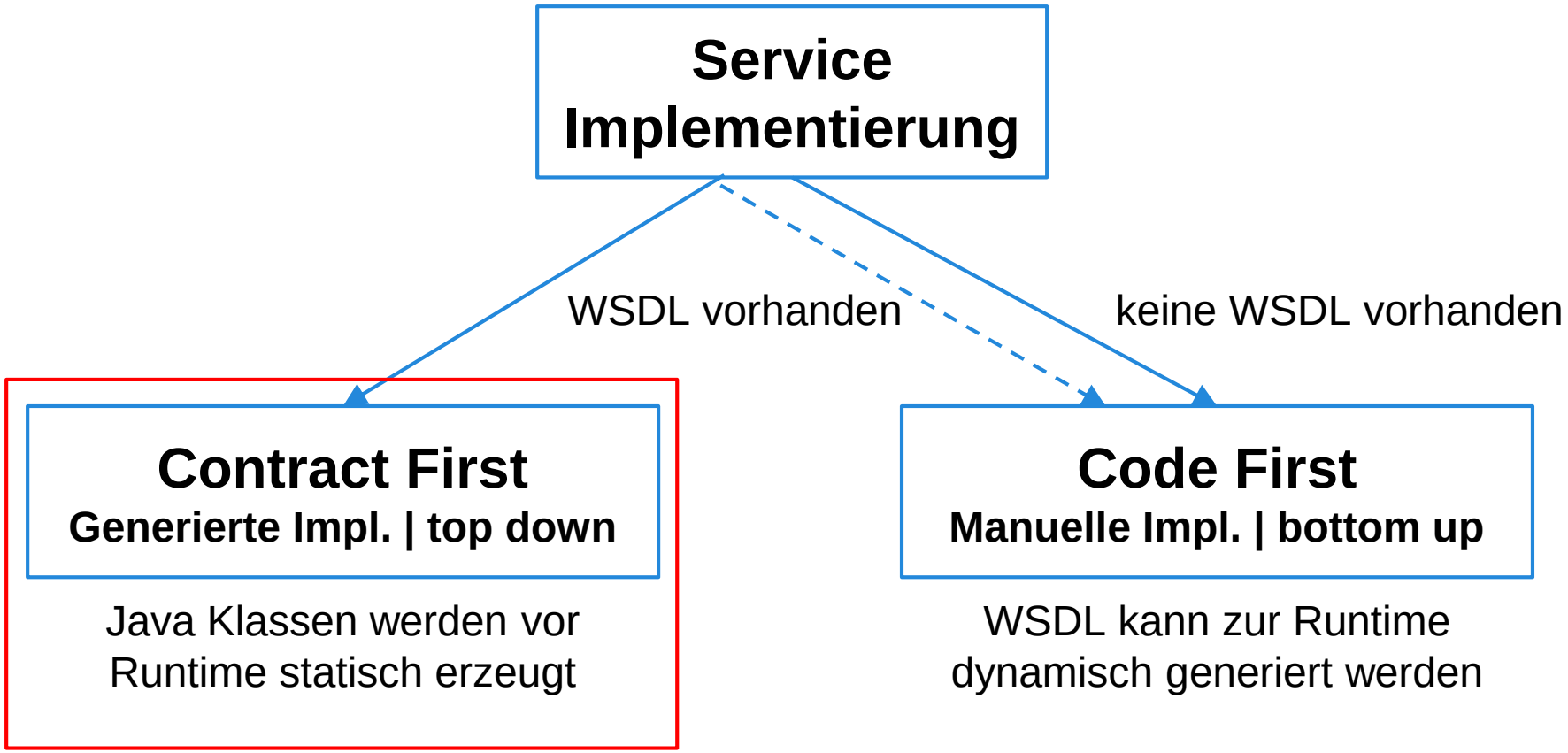
https://stackoverflow.com/questions/17553779/do-we-need-to-generate-stubs-while-writing-a-jax-ws-client

1. Stellt WSDL zur Verfügung
2. Client sendet SOAP-Request
3. Empfang und Zuweisung
4. Verarbeitet des SOAP-Message
5. Verarbeitet durch Handler
6. XML <> Java Typen Mapping bzw. Binding
7. Aufruf der Java Methode
8. Fehlerbehandlung (4. - 7.)
9. XML SOAP-Response an Client senden

Umsetzung


Service Implementierung
WSDL vorhanden
keine WSDL vorhanden
Contract First
Generierte Impl. | top down
Code First
Manuelle Impl. | bottom up
Java Klassen werden vor Runtime statisch erzeugt
WSDL kann zur Runtime dynamisch generiert werden

Umsetzung


Service Implementierung
WSDL vorhanden
keine WSDL vorhanden
Contract First
Generierte Impl. | top down
Java Klassen werden vor Runtime statisch erzeugt
Code First
Manuelle Impl. | bottom up
WSDL kann zur Runtime dynamisch generiert werden

Umsetzung - Contract First (1)

- Startet von WSDL
- wsimport generiert Interface, wie auf Client-Seite
  - $ wsimport -s src/main/java Calculator.wsdl
  - RI wie z.B. von Apache CXF wsdl2service

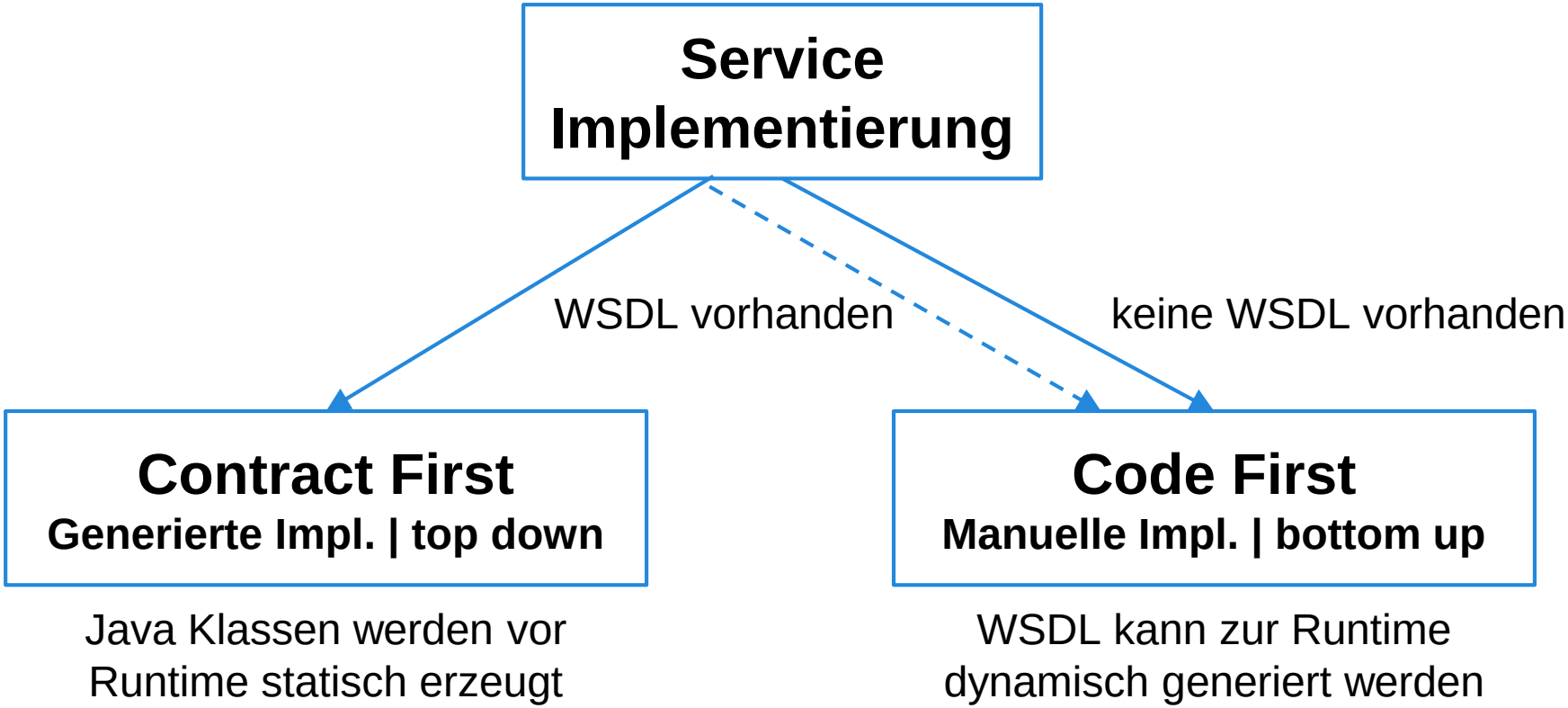
Umsetzung - Contract First (2)

- Manuelle Implementierung des SEI

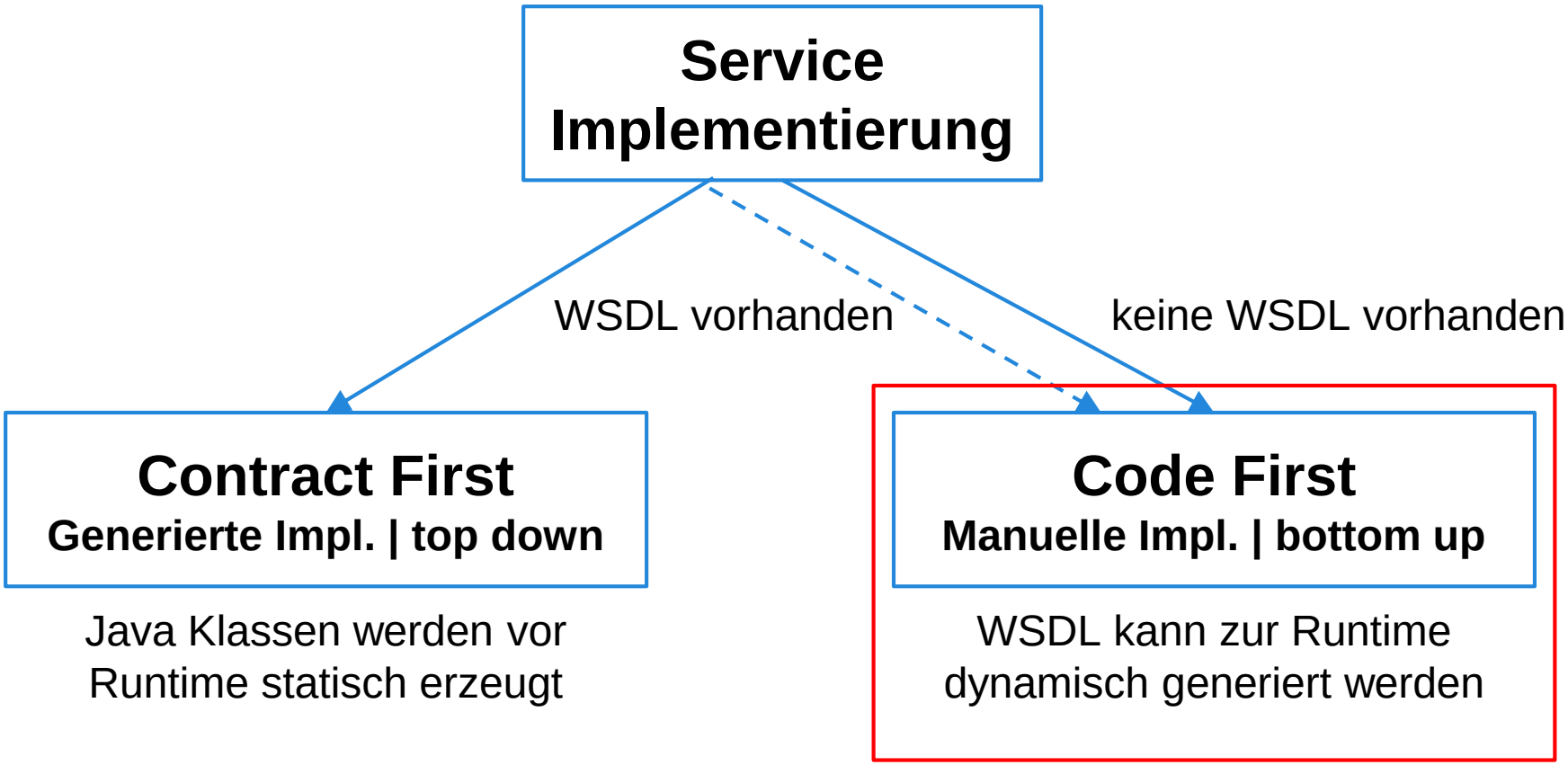
```
@WebService(…,wsdlLocation = "echo.wsdl")
public class Service implements EchoPort {
        @Override
        public SayHelloResponse sayHello(final String name) {
                ObjectFactory of = new ObjectFactory();
                SayHelloResponse response = of.createSayHelloResponse();
                // ...
                return response;
        }
}
```

- Java EE Container stellt den WebService-Endpunkt bereit

Umsetzung

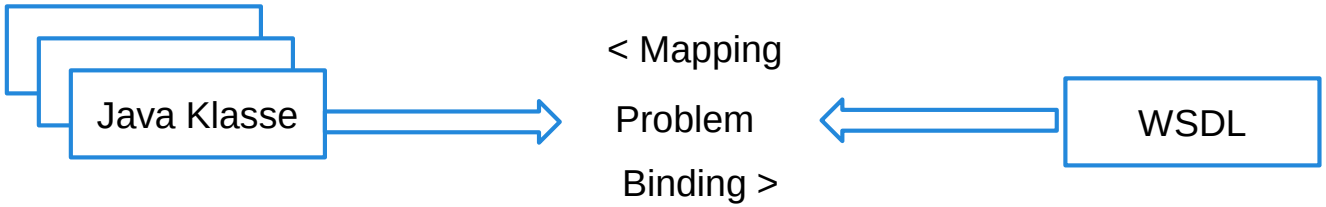

Service Implementierung
WSDL vorhanden
keine WSDL vorhanden
Contract First
Generierte Impl. | top down
Code First
Manuelle Impl. | bottom up
Java Klassen werden vor Runtime statisch erzeugt
WSDL kann zur Runtime dynamisch generiert werden

Umsetzung

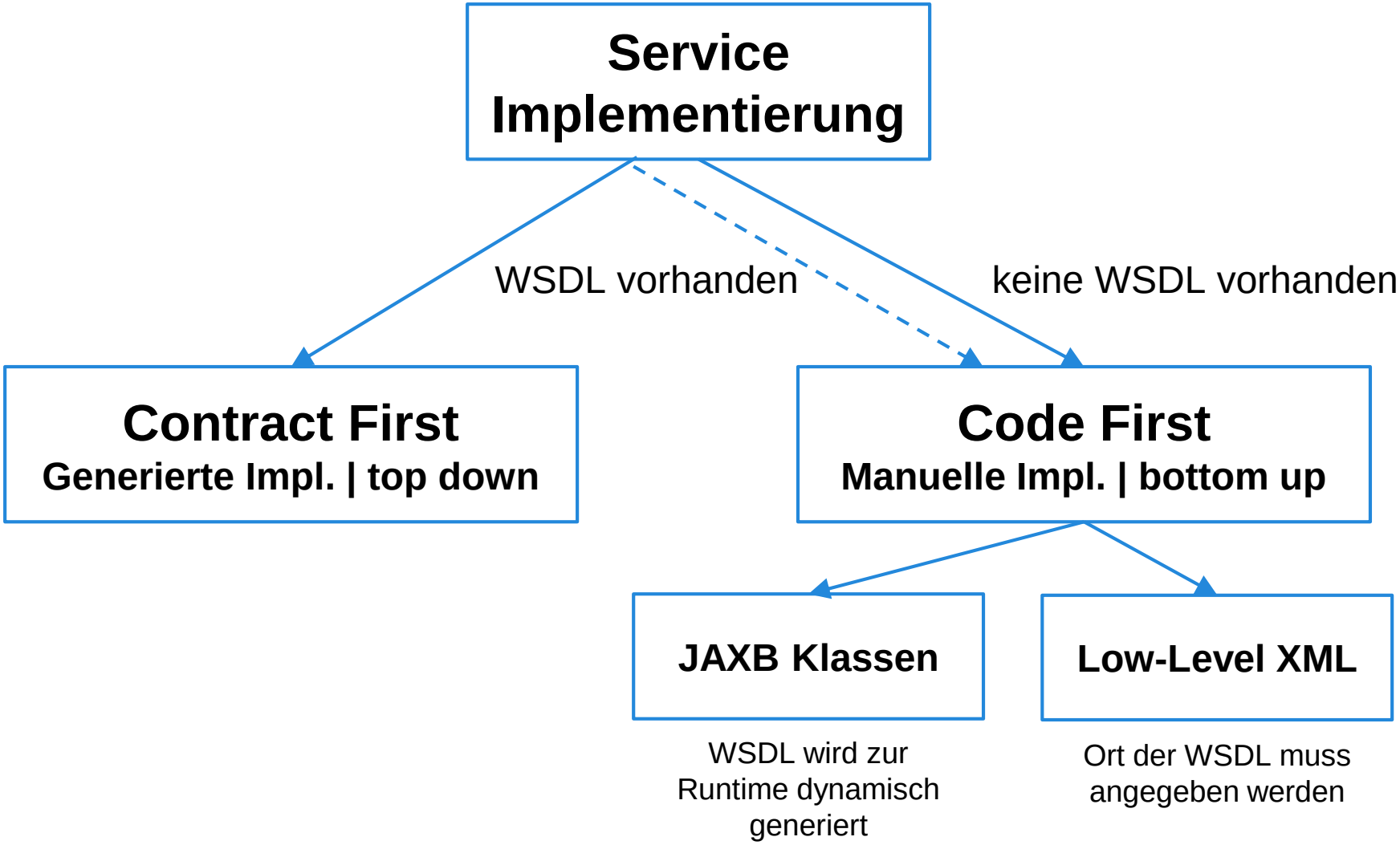

Service Implementierung
WSDL vorhanden
keine WSDL vorhanden
Contract First
Generierte Impl. | top down
Java Klassen werden vor Runtime statisch erzeugt
Code First
Manuelle Impl. | bottom up
WSDL kann zur Runtime dynamisch generiert werden

Umsetzung - Code First

- Start von Java
- WSDL wird von Java Klassen erzeugt
- Serviceanbieter muss WSDL zur Verfügung stellen (Webservice Endpunkt)
  - JAX-WS Runtime generiert diese automatisch (nur High-Level)
- Java <-> XML (JAXB)



Umsetzung


Service Implementierung
WSDL vorhanden
keine WSDL vorhanden
Contract First
Generierte Impl. | top down
Code First
Manuelle Impl. | bottom up
JAXB Klassen
Low-Level XML
WSDL wird zur Runtime dynamisch generiert
Ort der WSDL muss angegeben werden

Umsetzung - Code First - Low-Level

- XML <> Java Mapping muss manuell bzw. Low-Level umgesetzt werden
- Kein automatische Generierung der WSDL
  - Start von WSDL und Java
- Eignung nur für kleinere Serviceanbieter
- Source Code Ausschnitte
  - @WebServiceProvider(..., wsdlLocation = “...”)
  - class: implements Provider <SOAPMessage>
  - method: public SOAPMessage invoke(final SOAPMessage request)
- Veröffentlichung der SEI Impl.

Umsetzung - Code First - High-Level

- Implizite Nutzung von JAXB für das Mapping bzw. Binding (Un-/Marshalling)
- Annotaionen erzeugen alle Artefakte
- Minimaler Aufwand für einen vollständigen Webservice
  - Konfiguration über Annotationen
  - Convention over Configuration (Namensräume, WSDL/SOAP Styles, ...)
    - Restriktion: mit JAX-WS & JAXB sind nicht alle möglichen XML-Schematas realisierbar

Annotations

JSR 181

- @WebService
- @WebMethod
- @OneWay
- @WebParam
- @WebResult
- @HandlerChain
- @SOAPBinding

JSR 224

- @Action
- @BindingType
- @FaultAction
- @RequestWrapper
- @ResponseWrapper
- @ServiceMode
- @WebEndpoint
- @WebFault
- @WebServiceClient
- @WebServiceProvider
- @WebServiceRef

https://docs.oracle.com/cd/E13222_01/wls/docs103/webserv_ref/annotations.html

Implementierung

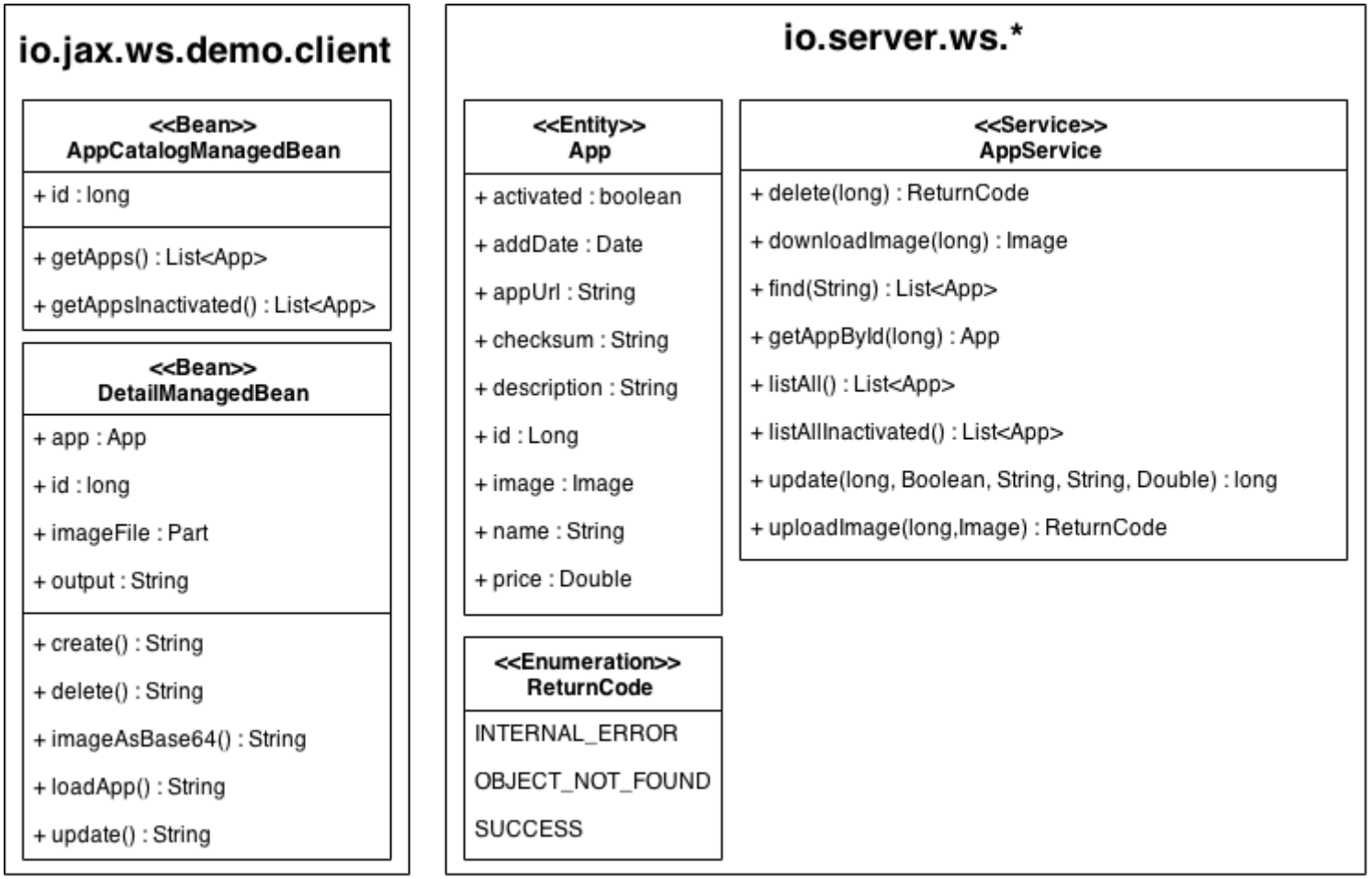
- Apache Axis
- Metro
- Apache CXF
- Spring WS
- Jersey
- und weitere

Live Demo - Samples

Let’s hack…

http://moviemezzanine.com/wp-content/uploads/movies-angelina-jolie-film-hackers-movie-2000x1309-wallpaper_www.wallpaperhi.com_6.jpg

Beispiel - AppStore



Live Demo - AppStore

Let’s hack…again

http://fc09.deviantart.net/fs71/i/2011/216/f/1/hackers_by_mp1331-d45glvk.png

Zusammenfassung

- JAX-WS Runtime auf Client und Server-Seite ab JDK 1.5
- Spezifikation JSR 224
- Annotations erleichtern Implementierung
- Codegenerator wsimport
- Handler zur Einflussnahme auf Nachrichten

Fazit: JAX-WS erleichtert erheblich die Implementierung von WebServices

Vielen Dank für die Aufmerksamkeit

Sourcecode & Quellen

Sourcecode:

- https://github.com/der-basti/jax-ws-demo

Quellen:

- Java Webservices; Friedrich Kiltz; 2010; mitp; Heidelberg
- Java Web Services in der Praxis; Heuser, Holubek; 2010; dpunkt.verlag; Heidelberg
- EJB 3.1 professional (2. Auflage); Ihns, Heldt, Koschek, Ehm, Sahling, Schlömmer; 2011; dpunkt.verlag; Heidelberg

Quellen

Quellen:

- https://jax-ws.java.net/
- https://cxf.apache.org/docs/
- https://docs.oracle.com/javase/6/docs/technotes/tools/share/wsimport.html
- https://www-01.ibm.com/support/knowledgecenter/SSAW57_7.0.0/com.ibm.websphere.nd.doc/info/ae/ae/welc6tech_wbs_dev.html?cp=SSAW57_7.0.0%2F1-12-37-0&lang=de
- http://dbis.eprints.uni-ulm.de/458/1/Klei08.pdf
- https://octodex.github.com/