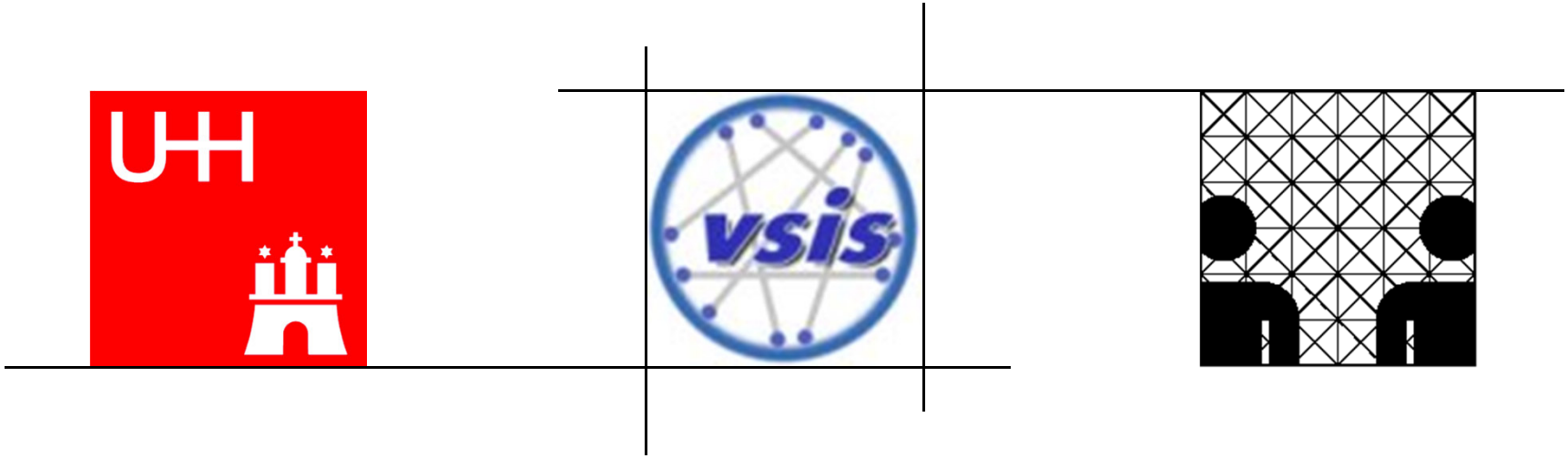


Vorlesung:

Komponenten-, service- und agentenorientierte Systeme

Service Oriented Architecture (SOA): Basistechnologien

Sommersemester 2015

Alexander Pokahr

Gliederung

Web Services Überblick

Standards

- SOAP, WSDL, UDDI, …

Technologien

- Web Service Programmierframeworks
- ESB

Web Services

Ziele

- Interoperabilität (Sprachunhabhängigkeit, Hersteller-unabhängigkeit)
- schnelle Einsetzbarkeit (adoption) (→ vgl. CORBA)
- dynamisches Binden ermöglichen (→ SOA)
- für offene und geschlossene Umgebungen

Anforderungen

- Standards und breite Unterstützung notwendig
- Voraussetzen von lediglich minimaler Infrastruktur
- nur geringe Kopplung zwischen Anwendungen
- Fokus auf *Nachrichten* und *Dokumente* (vgl. API)

Web Services: Definitionen

Ansatz zur Systemintegration nach dem Motto: *“alles neu, macht der Mai…”*:

„A web service is any service that is available over the Internet, uses a standardized XML messaging system, and is not tied to any one operating system or programming language.”
[Ethan Cerami, *Web Services Essentials,* O'Reilly, 2002]

“A Web service is a software system identified by a URI, whose public interfaces and bindings are defined and described using XML. Its definition can be discovered by other software systems. These systems may then interact with the Web service in a manner prescribed by its definition, using XML based messages conveyed by Internet protocols.”
[W3C, *Web Services Glossary*]

Web Services are „self-contained, modular business applications that have open, Internet-oriented, standards-based interfaces.”
[UDDI consortium: *UDDI_Executive_white_paper*, 2001]

Web Services: Charakteristika

Ein Web Service ist also ein Dienst, der…

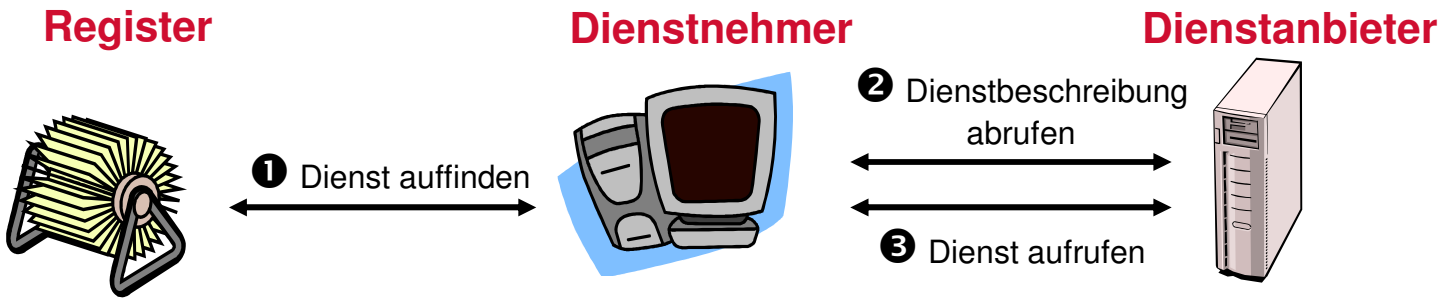
…über das *Internet* oder *Intranets* verfügbar ist,

…ein *standardisiertes (XML-) Nachrichtensystem* verwendet,

…programmiersprachen- und *betriebssystemunabhängig* ist,

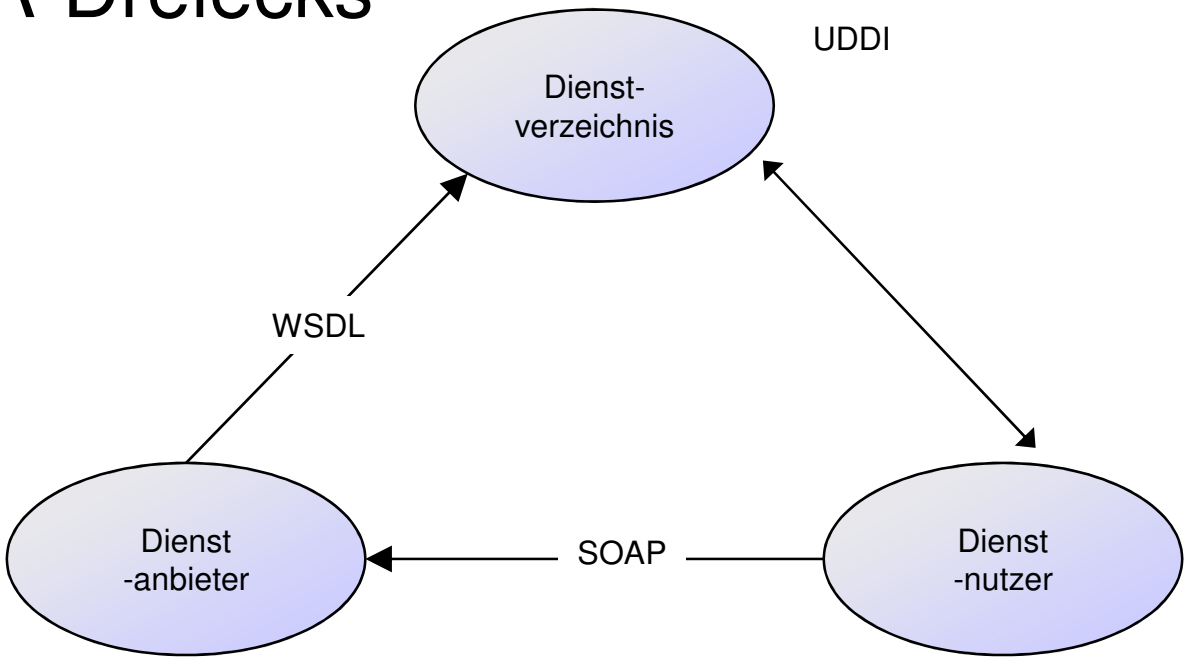
…unter Verwendung einer XML-Grammatik *selbstbeschreibend* ist

…über einen einfachen Mechanismus *auffindbar* ist.



Basistechnologien für Webservices

Webservice-Technologien als Inkarnation des SOA-Dreiecks



Web Services: Charakteristika (2)

Das hinter den *Web Services* stehende Konzept ist nicht neu – z.B. CORBA, DCE, DCOM etc. können das doch auch – aber:

bisherige Ansätze sind meist sehr komplexe Standards und

- sie benötigen eine neue / eigene Infrastruktur
- sie sind gut für die Server-zu-Server Kommunikation
- sie bieten untereinander keine Interoperabilität

Web Services verwenden die *standardisierten Internetprotokolle*, d.h.

- Firewalls und Proxies können somit leicht durchdrungen werden;
- es ist keine neue Infrastruktur ist notwendig – Erprobtes wird verwendet,
- sie sind gut für Client-to-Server Kommunikation und
- sie sind untereinander voll interoperabel.

Gliederung

Web Services Überblick

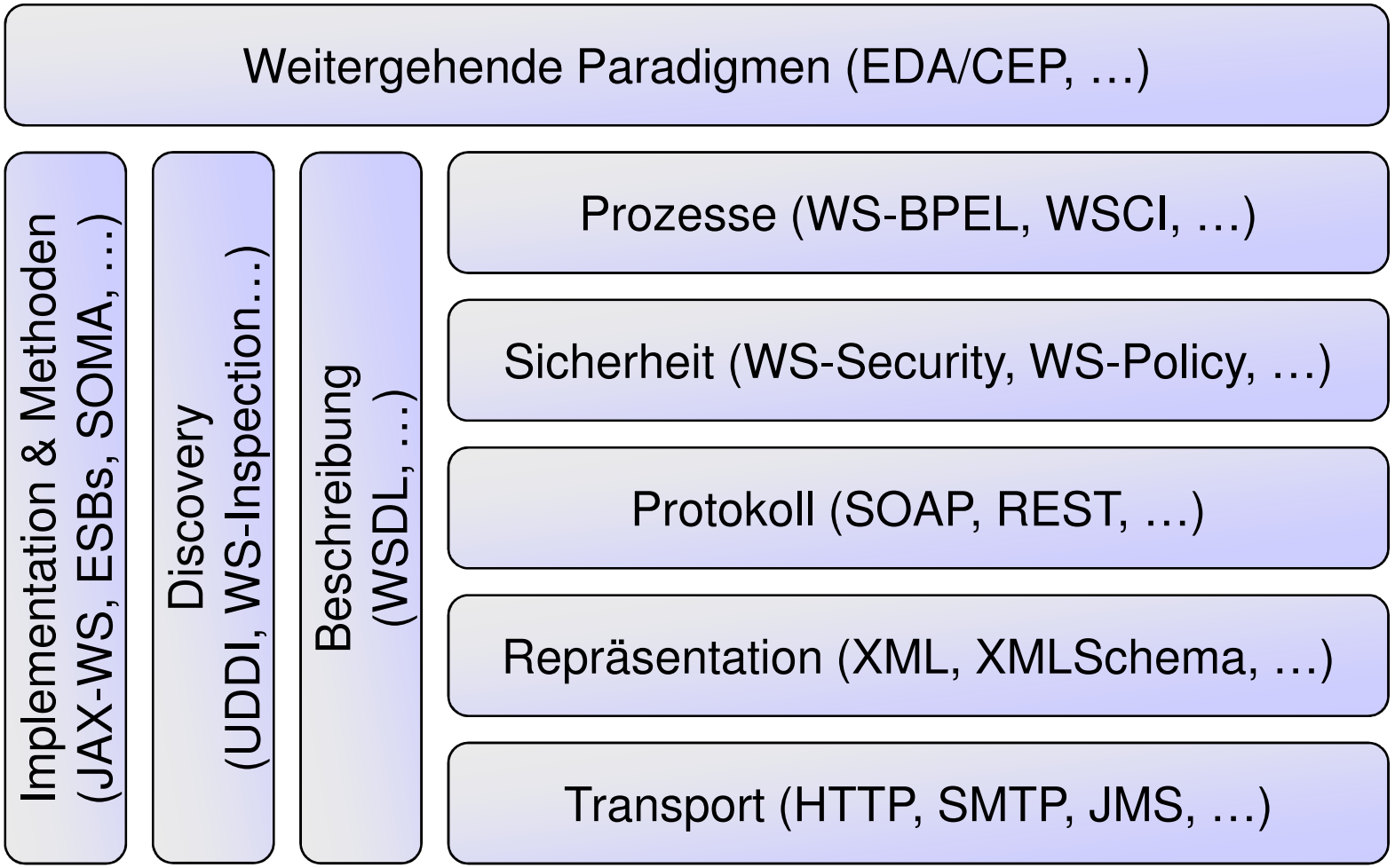
Standards

- **SOAP, WSDL, UDDI, …**

Technologien

- Web Service Programmierframeworks
- ESB

Webservices “Big Picture”

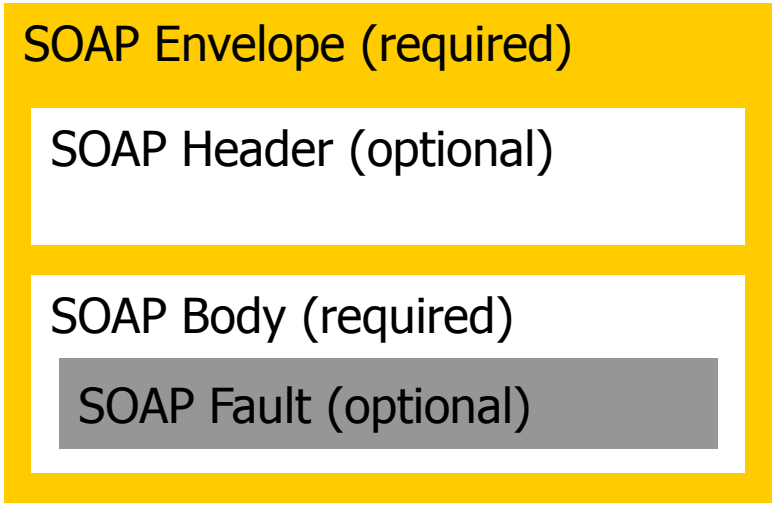


WS Messaging: SOAP

plattformunabhängiges, zustandsloses und XML-basiertes Protokoll zum Austausch von Einwegnachrichten

komplexere Kommunikationsformen können auf Basis dieser Einwegnachrichten erzeugt werden (request/response etc.)

SOAP-Nachrichten bestehen aus einem „Umschlag“, einem optionalen „Kopfteil“ und einem „Nachrichtenkörper“ mit optionaler Fehlerbehandlung



SOAP Beispiel

Header erlaubt Zwischenstationen für die Verarbeitung

```
<SOAP:Envelope xmlns:SOAP="http://schemas.xmlsoap.org/soap/envelope”
  SOAP:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
   <SOAP:Header>
      <a:authentication role="intermediary a"…>…</a:authentication>
      <s:security role="intermediary b"…> … </s:security>
      <t:transactions actor="intermediary c"…> … </t:transactions>
      <p:payment actor=“ultimateReceiver"…> … </p:payment>
   </SOAP:Header>
   <SOAP:Body>
      <m:mybody> … </m:mybody>
   </SOAP:Body>
</SOAP:Envelope>
```

WS Messaging: REST

Alles ist eine Ressource

- Alle Operationen der Anwendung müssen als Zugriff auf eine Ressource modelliert werden

Zugriff auf Ressourcen über HTTP (zustandslos)

- GET: Eine Ressource abfragen
- POST: Ressource anlegen (neue ID)
- PUT: Ressource ändern (alte ID)
- DELETE: Eine Ressource löschen
- HTTP-Fehlercodes als Ergebnis (200=OK, 404=Not Found, …)

Beispiel: Bestellung im Online-Shop

- POST: Bestellung aufgeben
- GET: Status der Bestellung abfragen
- PUT: Bestellung ändern
- DELETE: Bestellung stornieren

Messaging: Zusammenfassung

REST

- einfache aber eingeschränkte Metapher
- Verwendung von HTTP-Aufrufen (GET/POST/ etc.)

SOAP

- nachrichtenbasiertes Modell
- Meta-Informationen zu Nachrichten im Header
- Verarbeitung auch über Zwischenstationen
- Fehlerbehandlung

WS Beschreibung: WSDL

WSDL = Web Services Description Language

- XML-Grammatik zur Beschreibung von Web Services
- unabhängig von der verwendeten Plattform und Programmiersprache

WSDL dient zur funktionalen Beschreibung eines (Software-) Dienstes

- IDL zur Beschreibung (standardisierter) Dienstschnittstellen
- ermöglicht Service Advertisements
- ermöglicht Tool-Support (z.B. Generierung von Client- und Server-Stubs)

WSDL-Beschreibung enthält

- plattformunabhängige Beschreibung des Dienstes (abstrakte Schnittstelle)
- Details zu Protokollen und Deployment (konkrete Bindings)

Standardisierung durch das W3C

- aktuelle Version: WSDL 2.0 (vorher 1.2)
- WSDL 1.1 noch im Einsatz verbreitet

WSDL in a nutshell…

`<definitions>`		Wurzelelement in WSDL
`<types>`		Welche Datentypen werden übertragen ?
`<messages>`		Welche Nachrichten gibt es ?
`<portType>` `<interface>`	(1.1) (2.0)	Welche Operationen/Funktionen werden unterstützt ?
`<binding>`		Welches Transportprotokoll wird verwendet und wie sind dessen Details ?
`<service>`		Wo finde ich den Web Service ?

WSDL Zusammenfassung

WSDL entkoppelt entfernte Dienstaufrufe

- Types = anwendungsspezifische Nachrichteninhalte
- Message = Nachrichtentypen für die Kommunikation

- PortTypes = Dienstschnittstellen (Interfaces in WSDL 2.0)
- Operations = Methodensignaturen

- Bindings = Abbildung auf technische Kommunikations-infrastruktur
- Ports = Adressen der Dienstimplementation

WSDL: Aufbau

Service

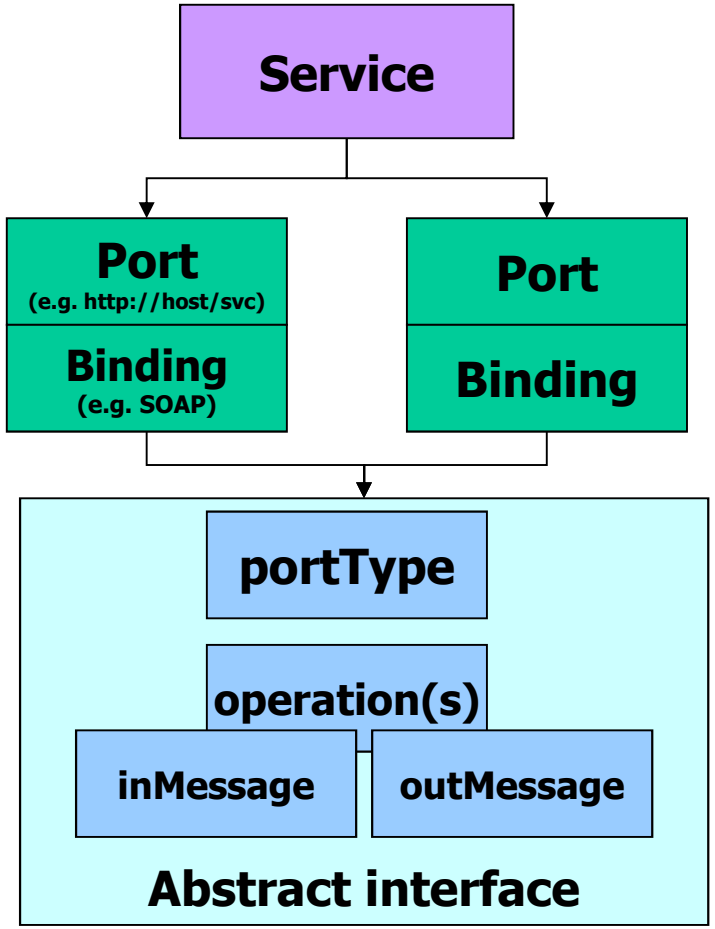
- konkrete Dienstbeschreibung
- verschiedene Bindings je portType (SOAP, JMS, …)

Port

- (eine) Adresse des Dienstes

portType / interface

- abstrakte Dienstbeschrei-bung
- Besteht aus Menge von Operationen



Web Service Discovery: UDDI

Universal Description, Discovery, and Integration

- definiert, wie Informationen über Web Services und deren Anbieter publiziert und gefunden werden können
- umfasst technische Schnittstelle sowie Nutzungsregeln für globales Repository

UDDI besteht aus drei Elementen:

- einem logisch zentralen, physikalisch aber verteiltem Registry-Service,
- XML-Schemata zur Beschreibung der Anbieter und ihrer Web Services und
- einer SOAP-Schnittstelle für den Zugriff auf die Registry

Datenstrukturen für die Registrierung von

- Unternehmen (businesses)
- Spezifikationen von Dienstarten
- Diensten und Endpoints

UDDI enthält Informationen aus drei Kategorien:

White Pages	**Yellow Pages**	**Green Pages**
Allgemeine Anbieterdaten abgelegt nach Namen	Inhalt abgelegt nach Kategorien (z.B. Dienstarten)	Technische Informationen der angebotenen Dienste

UDDI Registrierung

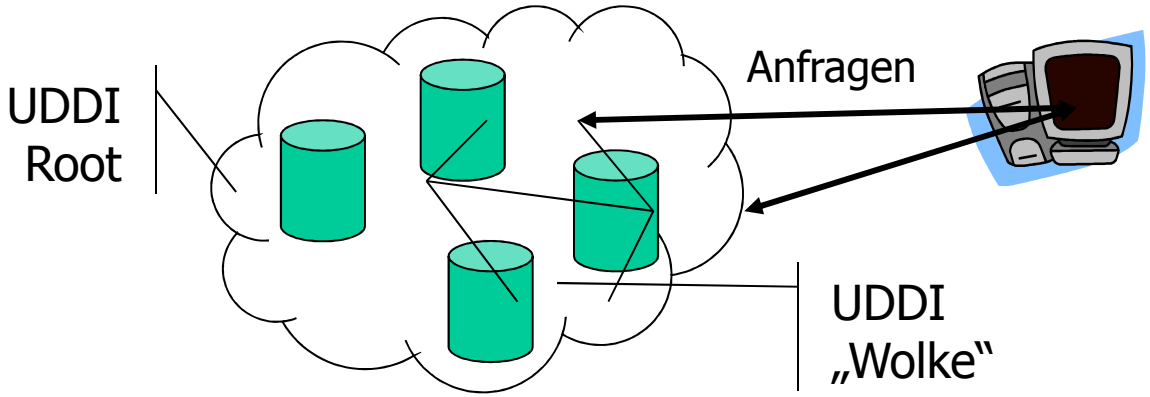
Die ***UDDI Registry*** ist logisch zentralisiert, physikalisch aber verteilt realisiert

Eine Replikation der Daten unter den einzelnen Knoten erfolgt alle 24 Stunden

- Alle Knoten enthalten jeweils alle registrierten Daten

Registrierungen können entweder über eine WWW-Schnittstelle oder über die API vorgenommen werden

- Die SOAP-Schnittstelle wird von allen Knoten unterstützt



Alternativen zu UDDI

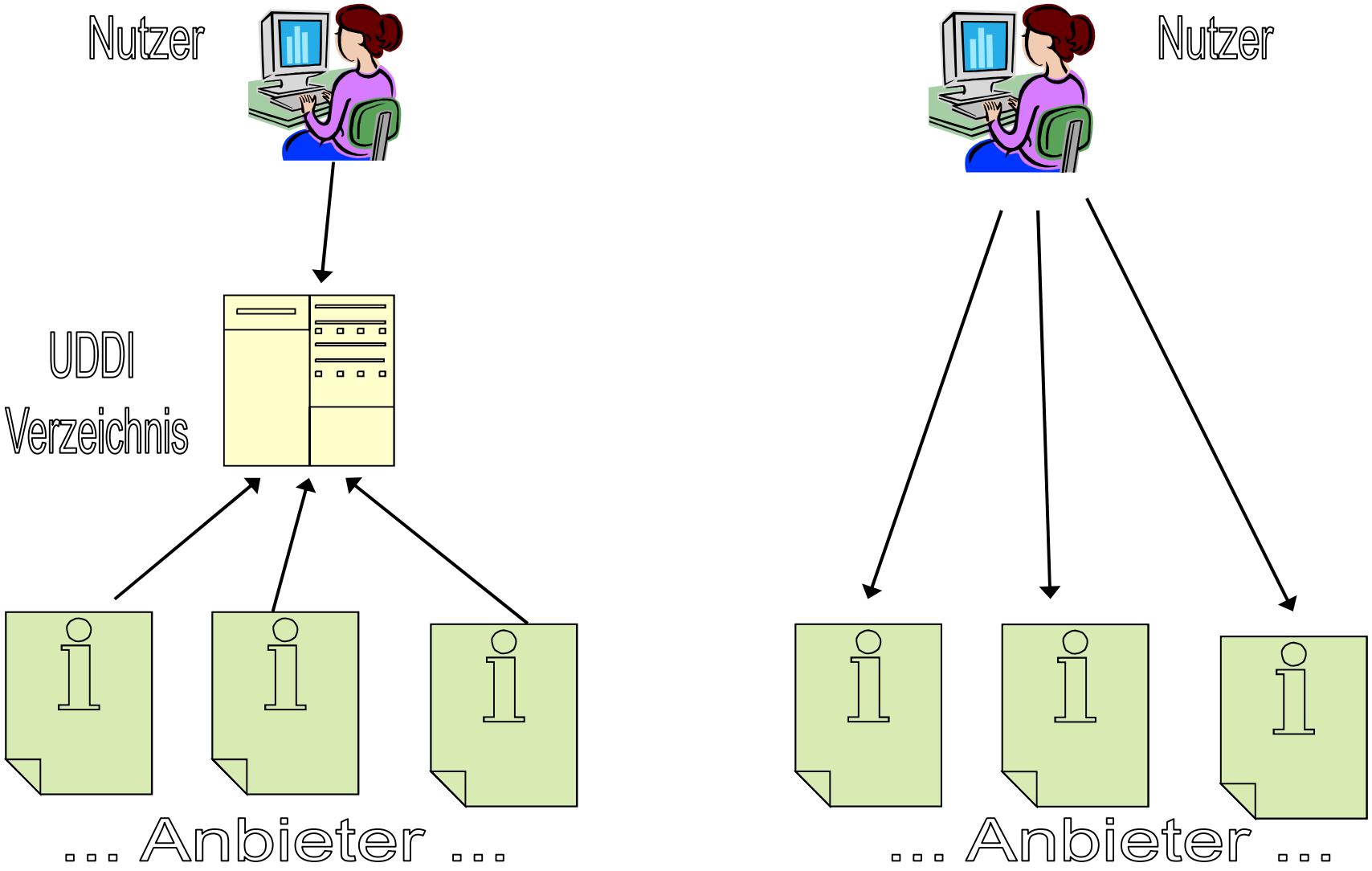
WS-Inspection

- WSDL-Beschreibungen werden vom Dienstanbieter (provider) selbst zur Verfügung gestellt
- einfache Publikation/Aktualisierung
- Suche?

WS-Discovery

- dynamisches Auffinden neuer Services (Multicast-basiertes Discovery-Protocol)
- auch für Ad-hoc-Netzwerke geeignet
- begrenzte Skalierbarkeit (Internet?)

UDDI vs. WS-Inspection/Discovery



Gliederung

Web Services Überblick

Standards

- SOAP, WSDL, UDDI, …

Technologien

- **Web Service Programmierframeworks**
- ESB

Web-Service Frameworks Überblick

Name	Platform	Messaging Model(Destination)	Specifications	Protocols
Apache Axis	Java/C++	Client/Server	WS-ReliableMessaging, WS-Coordination, WS-Security, WS-AtomicTransaction, WS-Addressing	SOAP, WSDL
Apache Axis2	Java	Client/Server/ Asyn Support	WS-ReliableMessaging, WS-Security, WS-AtomicTransaction, WS-Addressing, MTOM, WS-Policy, WS-MetadataExchange	SOAP1.1, SOAP1.2, MTOM, WSDL 2.0, WSDL, REST
Apache CXF	Java	Client/Server/ Asyn Support	WS-ReliableMessaging, WS-Security, WS-Addressing, MTOM, WS-Policy, WS-SecureConversation, WS-SecurityPolicy, WS-Trust	SOAP1.1, SOAP1.2, MTOM, WSDL 2.0, WSDL, REST
CodeIgniter	PHP	Client/Server	An open source MVC web application framework	XML-RPC
gSOAP	C and C++	Client/Server Duplex/Async	WS-Addressing, WS-Discovery, WS-Policy, WS-ReliableMessaging, WS-Security, WS-SecurityPolicy	SOAP1.1, SOAP1.2, MTOM, WSDL 1.1, WSDL 2.0, REST, XML-RPC, JSON, JSON-RPC, XML
Java Web Services Development Pack / GlassFish	Java	Client/Server	WS-Addressing, WS-Security, ???	SOAP, WSDL, ???
.NET Framework	C#, VB.NET	Client/Server	WS-Addressing, WS-MetadataExchange, WS-Security, WS-Policy, WS-SecurityPolicy, WS-Trust, WS-SecureConversation, WS-ReliableMessaging, WS-Coordination, WS-AtomicTransaction	SOAP, WSDL, MTOM
Web Services Interoperability Technology	Java	Client/Server	WS-Addressing, WS-ReliableMessaging, WS-Coordination, WS-AtomicTransaction, WS-Security, WS-Security Policy, WS-Trust, WS-SecureConversation, WS-Policy, WS-MetadataExchange	SOAP, WSDL, MTOM, JSON, XML
Web Services Invocation Framework	Java	Client	???	SOAP, WSDL
Windows Communication Foundation	.NET	Client/Server/Asyn support	WS-Addressing, WS-MetadataExchange, WS-Security, WS-Policy, WS-SecurityPolicy, WS-Trust, WS-SecureConversation, WS-ReliableMessaging, WS-Coordination, WS-AtomicTransaction	SOAP, WSDL, REST
WSO2 WSF/PHP	PHP	Client/Server	SOAP MTOM, WS-Addressing, WS-Security, WS-SecurityPolicy, WS-Secure Conversation, WS-ReliableMessaging	SOAP, WSDL
XFire *became Apache CXF*	Java	Client/Server	WS-Addressing, WS-Security	SOAP, WSDL
XML Interface for Network Services	Java	Server ?	??	SOAP, XML-RPC, WSDL, JSON-RPC, JSON
Zend Framework	PHP	Client/Server	?	SOAP, JSON, JSON-RPC, REST, XML-RPC

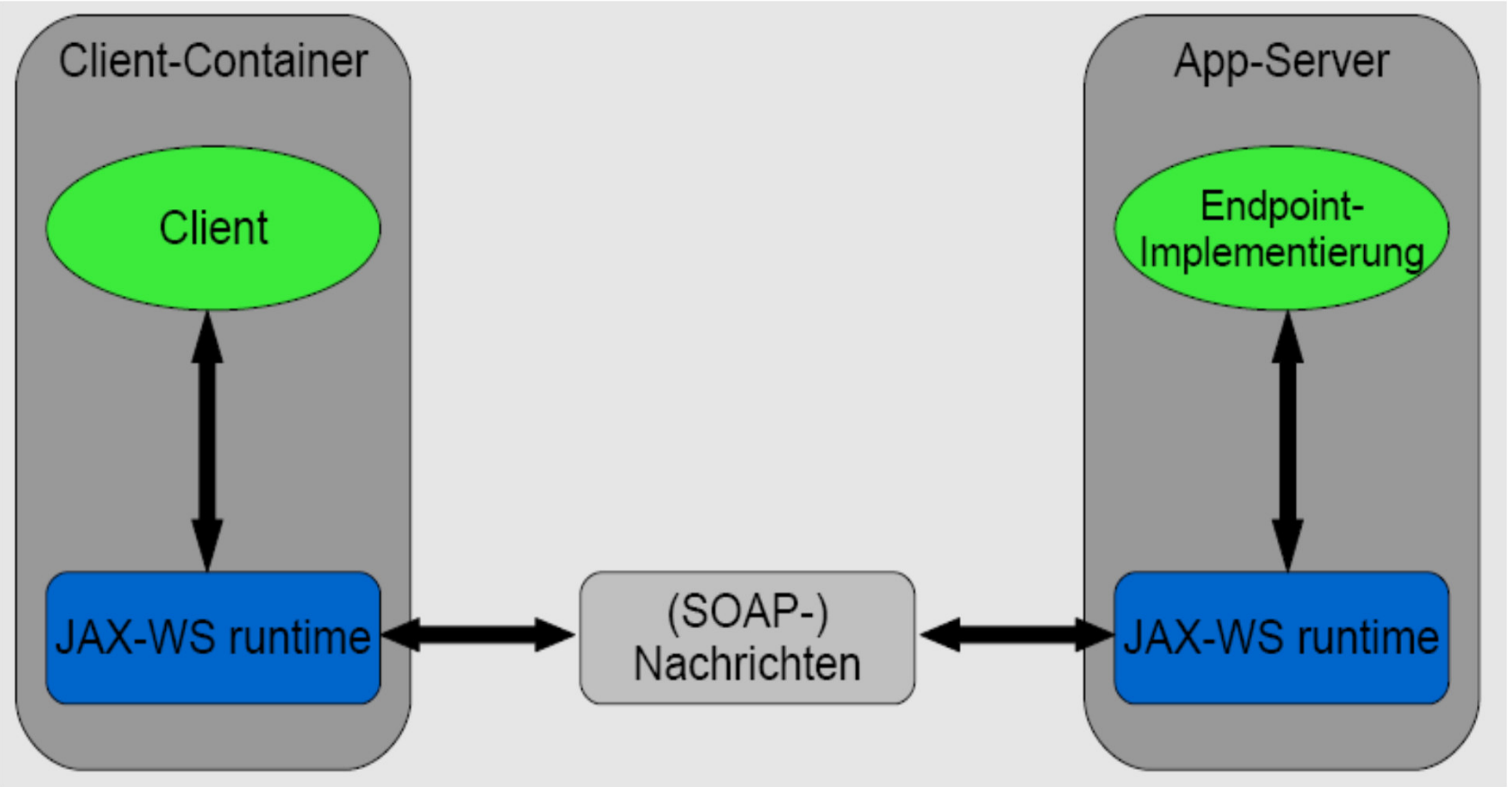
(Wikipedia 2014)

JAX-WS Webservice Implementation API

- Webservice API der Plattform Java EE 5 bzw. Java SE 6 für die Web Service-Entwicklung mit Java und XML
- Aktuelle Version JAX-WS 2.2
- Referenzimplementierung im Projekt Glassfish

- Vereinfachtes Programmiermodell durch Separierung von high- und low-level API
- Abstrakte Sicht auf Webservices ohne Details von SOAP, XML usw. (möglich)

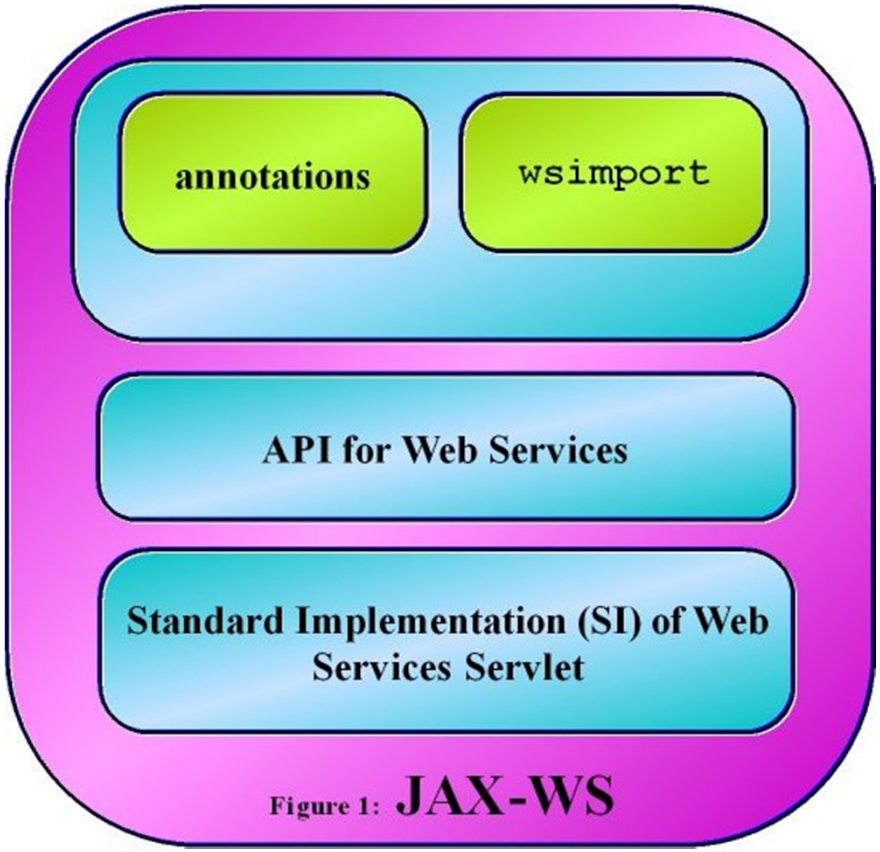
JAX-WS Aufrufinfrastruktur



(Keim 2007)

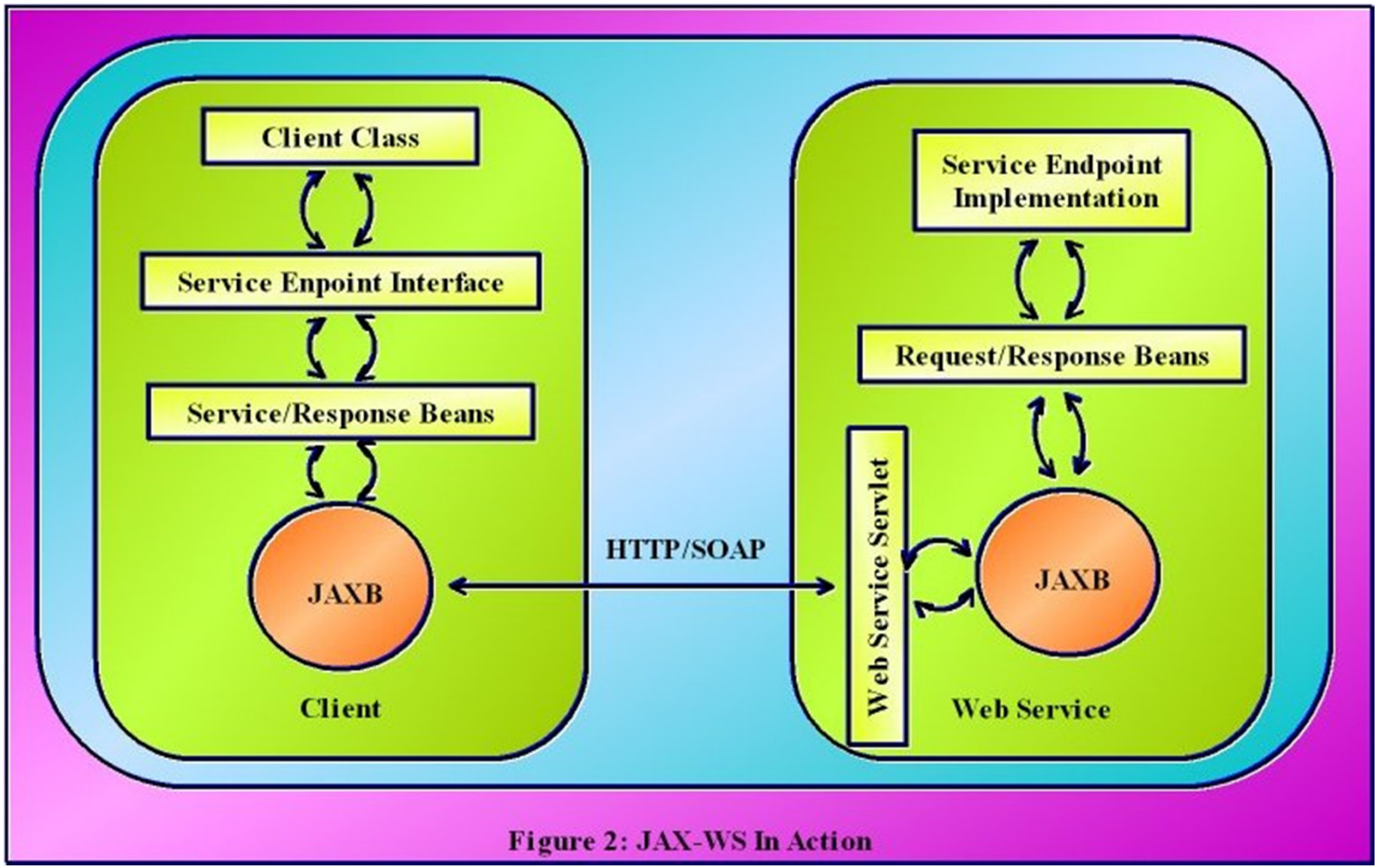
JAX-WS Bestandteile

- Beschreibung von Webservices mit Hilfe von Annotationen wie z.B. @WebMethod
- Standardimplementation (SI, wie im JAXWS-SI.jar) des Webservices Servlet
- Eine Bibliothek zur Konvertierung zwischen Java-Beans und XML-Elementen
- Tools zur Generierung von Webserviceartefakten wie z.B. Beans, Stubs, WSDL



(Yates 2006)

JAX-WS Aufrufsemantik verfeinert



(Yates 2006)

Entwicklung der Serverseite

Bottom-up:

1. Implementierung des Service-Endpoint Interfaces (SEI)
2. Generieren von Portable Artifacts (WSDL, XSD) aus dem SEI (Tool: wsgen)
3. Deployment als .war-Archiv

Top-down:

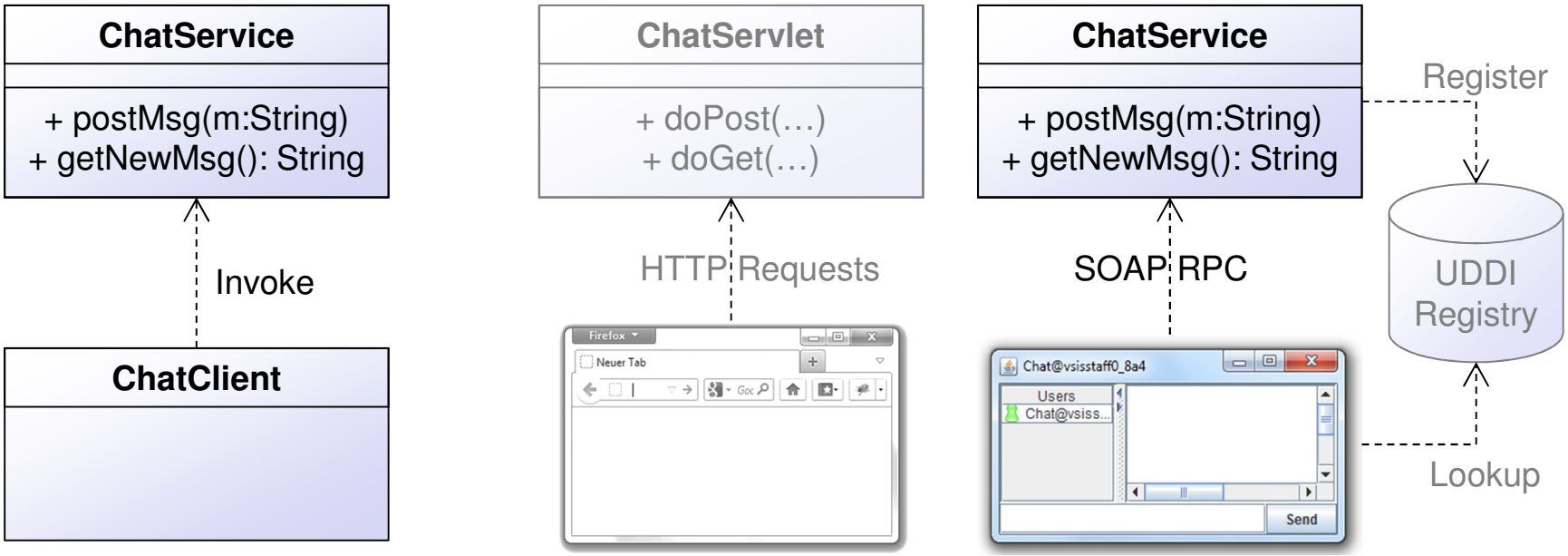
1. Beschreibung des Webservices als WSDL-Datei
2. Generieren von Portable Artifacts (Proxy-Klassen) aus WSDL und XSD (Tool: wsimport)
3. Implementierung des Service-Endpoint Interfaces (SEI)
4. Deployment als .war-Archiv

Clientseitige Implementierung

- Low-level API über Dispatch-Client:
  - Aufrufen von Webdiensten ohne typsicheren Proxy
  - Aufruf erfolgt auf XML-Nachrichtenebene ohne generierte Artefakte

- High-level API über Dynamic Proxy:
  1. Generieren von Portable Artifacts aus einer WSDL-Beschreibung (Tool: wsimport)
  2. Verwenden der generierten Proxy-Klassen zum Aufrufen des Webservices von einem beliebigen Endpunkt
     - Erzeuge Dienst
     - Hole Port-Proxy
     - Rufe Aktion im Port-Proxy auf

Beispiel: Chat



Bottom-up-Beispiel Serverseite

```java
@WebService
@SOAPBinding(style=Style.RPC)
public class ChatServer
{
    String  message;

    public void postMessage(String m)
    {
        this.message    = m;
        synchronized(this)
        {
            this.notifyAll();
        }
    }

    public String   getNewMessage()
    {
        synchronized(this)
        {
            try
            {
                this.wait();
            }
            catch(InterruptedException e)
            {
            }
        }
        return message;
    }
}
```

```java
public static void main(String[] args)
{
    ChatServer server = new ChatServer();
    Endpoint endpoint = Endpoint.publish("http://localhost:8080/chat", server);
    System.out.println("Server startet: "+endpoint);
    // Access WSDL from http://localhost:8080/chat?wsdl
    // Generate client classes with:
    // wsimport -keep -p wsdlchat.gen http://localhost:8080/chat?wsdl
}
```

Bottom-up Beispiel Clientseite

```
public class ChatClient
{
    public ChatClient(String name)
    {
        ChatServerService service = new ChatServerService();
        final wsdlchat.gen.ChatServer chat = service.getChatServerPort();
        final ChatGui   gui = new ChatGui(name)
        {
            public void sendMessage(String m)
            {
                chat.postMessage(m);
            }
        };

        new Thread(new Runnable()
        {
            public void run()
            {
                while(true)
                {
                    String  msg = chat.getNewMessage();
                    if(msg!=null)
                    {
                        gui.addMessage(msg);
                    }
                }
            }
        }).start();
    }

    public static void  main(String[] args) throws Exception
    {
        new ChatClient(args.length==0? "Noname": args[0]);
    }
}
```

Gliederung

Web Services Überblick

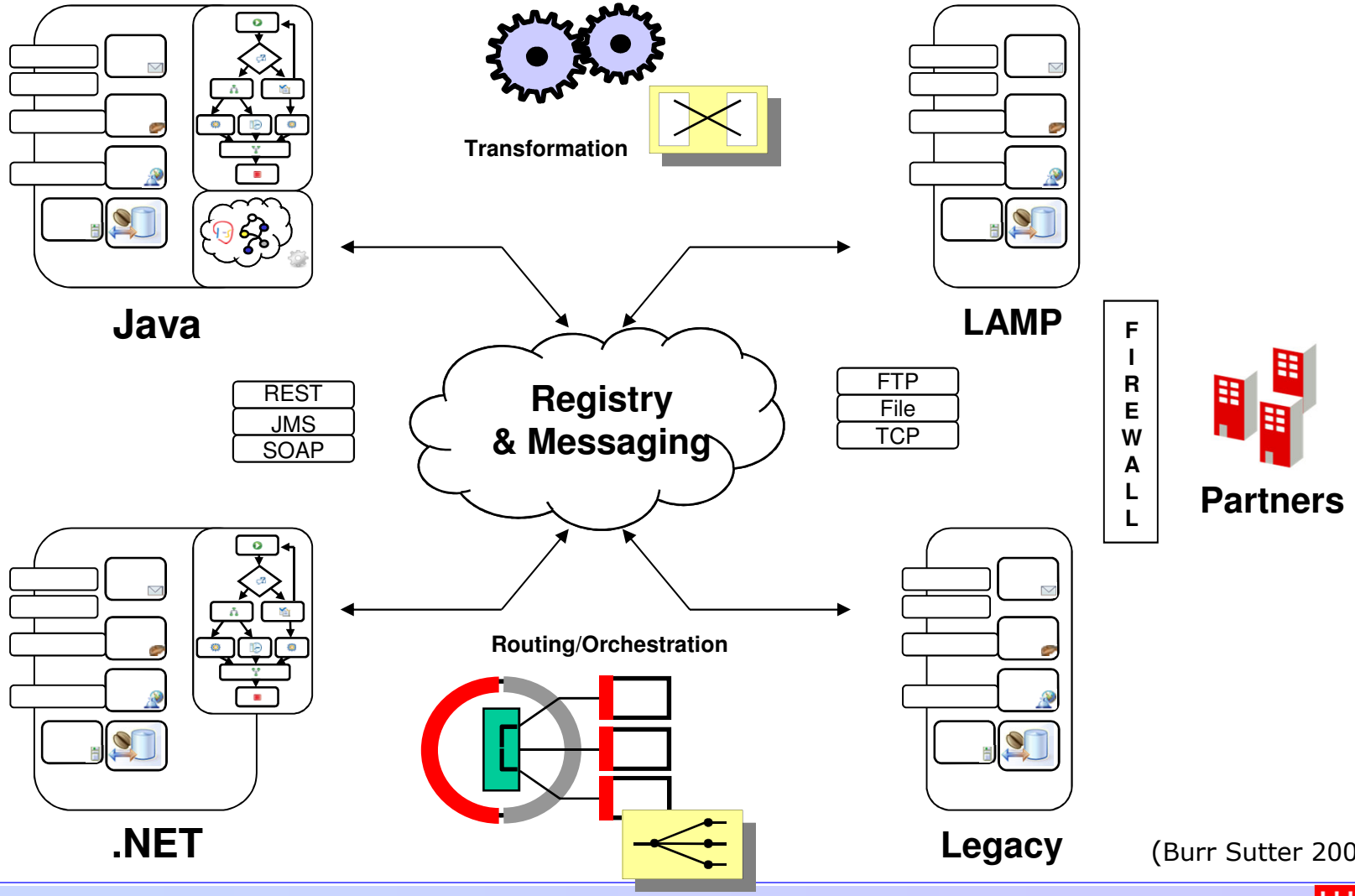
Standards

- SOAP, WSDL, UDDI, …

Technologien

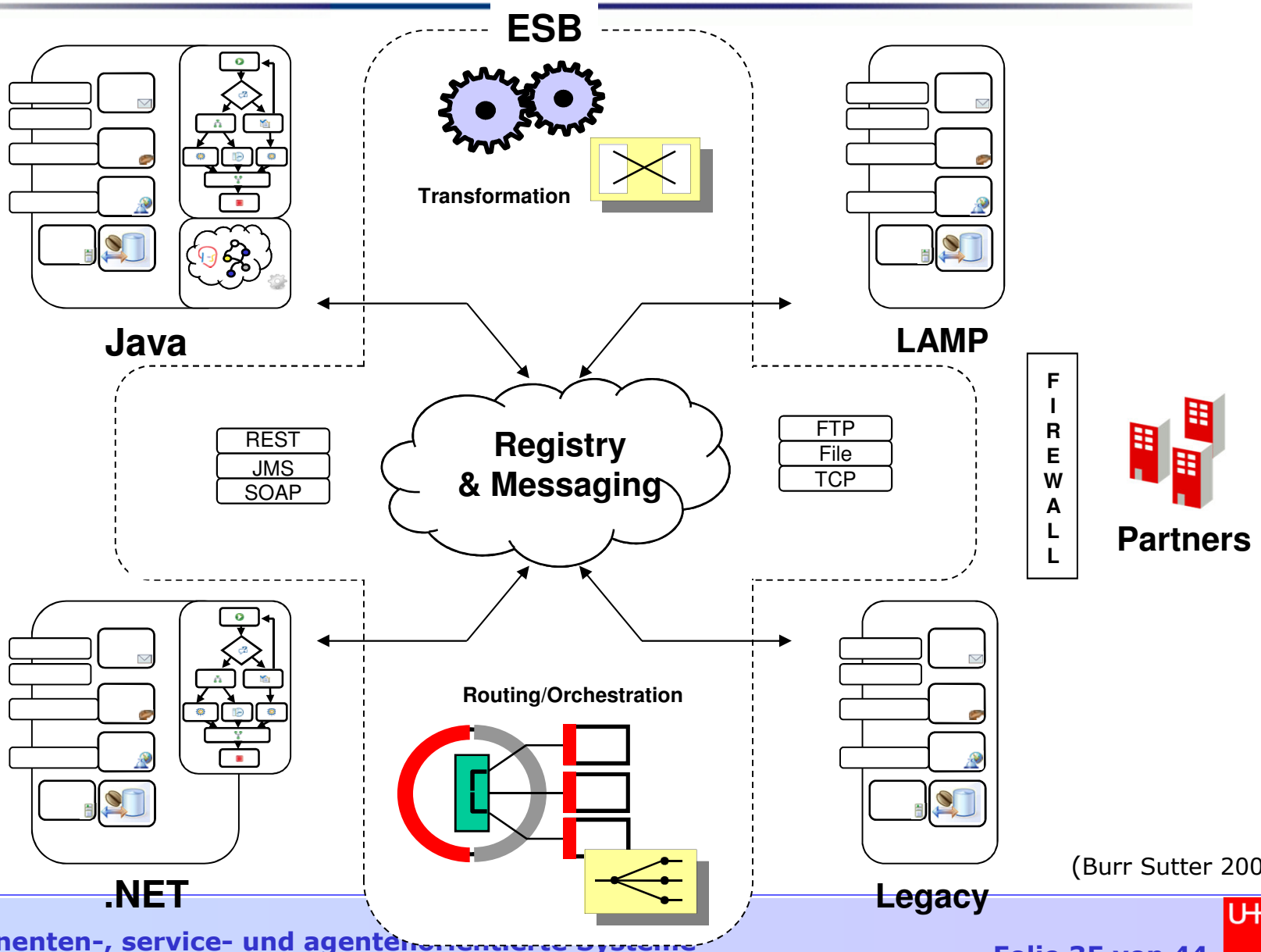
- Web Service Programmierframeworks
- **ESB**

App Server vs ESB for SOA



(Burr Sutter 2008)

App Server vs ESB for SOA



(Burr Sutter 2008)

Was ist ein ESB?

Keine Allgemeine Definition

- Architekturmuster vs. Produkt vs. Software- o. Hardwarekomponenten
- Wikipedia: „An enterprise service bus (ESB) is a software architecture model used for designing and implementing communication between mutually interacting software applications in a service-oriented architecture (SOA). “
- Gartner: „An Enterprise Service Bus (ESB) is a new architecture that exploits Web Services, messaging, middleware, intelligent routing, and transformation. ESBs act as a lightweight, ubiquitous integration backbone through which software services and application components flow“

ESB Fähigkeiten bzw. Aufgaben

- Routing
- Message Transformation
- Message Enhancement
- Protocol Transformation
- Service Mapping
- Message Processing
- Service Orchestration
- Transaction Management
- Security

(Kandaswamy 2009)

ESB Fähigkeiten bzw. Aufgaben (2)

Routing

- Aufrufe an geeignete Service Provider weiterleiten
- Statisch vs. Dynamisch
- Content-based, policy-based, complex rule-based

Message Transformation

- Struktur- und Formatanpassungen zur Entkopplung von Client und Provider
- Z.B.: XML → XML, XML → JSON, …

Message Enhancement

- Inhaltsanpassungen und hinzufügen zusätzlicher Daten
- Z.B.: Datumsformatkonvertierung, Hinzufügen von IDs

(Kandaswamy 2009)

ESB Fähigkeiten bzw. Aufgaben (3)

Protocol Transformation

- Vermittlung zwischen Transportprotokollen
- Z.B.: SOAP/HTTP, RMI/IIOP, …

Service Mapping

- Vom logischen Business Service zur konkreten Implementation
- Z.B.: Name → WSDL Binding

Message Processing

- Transaktionale Nachrichtenzustellung
- Vgl. MOM

(Kandaswamy 2009)

ESB Fähigkeiten bzw. Aufgaben (4)

Service Orchestration

- Ausführung von Geschäftsprozessen
- Beschrieben z.B. in WS-BPEL, BPMN 2.0

Transaction Management

- Verteilte Transaktionen über mehrere Services
- Z.B. nach WS-Coordination

Security

- Durch ESB haben Services höhere Sichtbarkeit
- Authentication, Authorization, Auditing benötigt

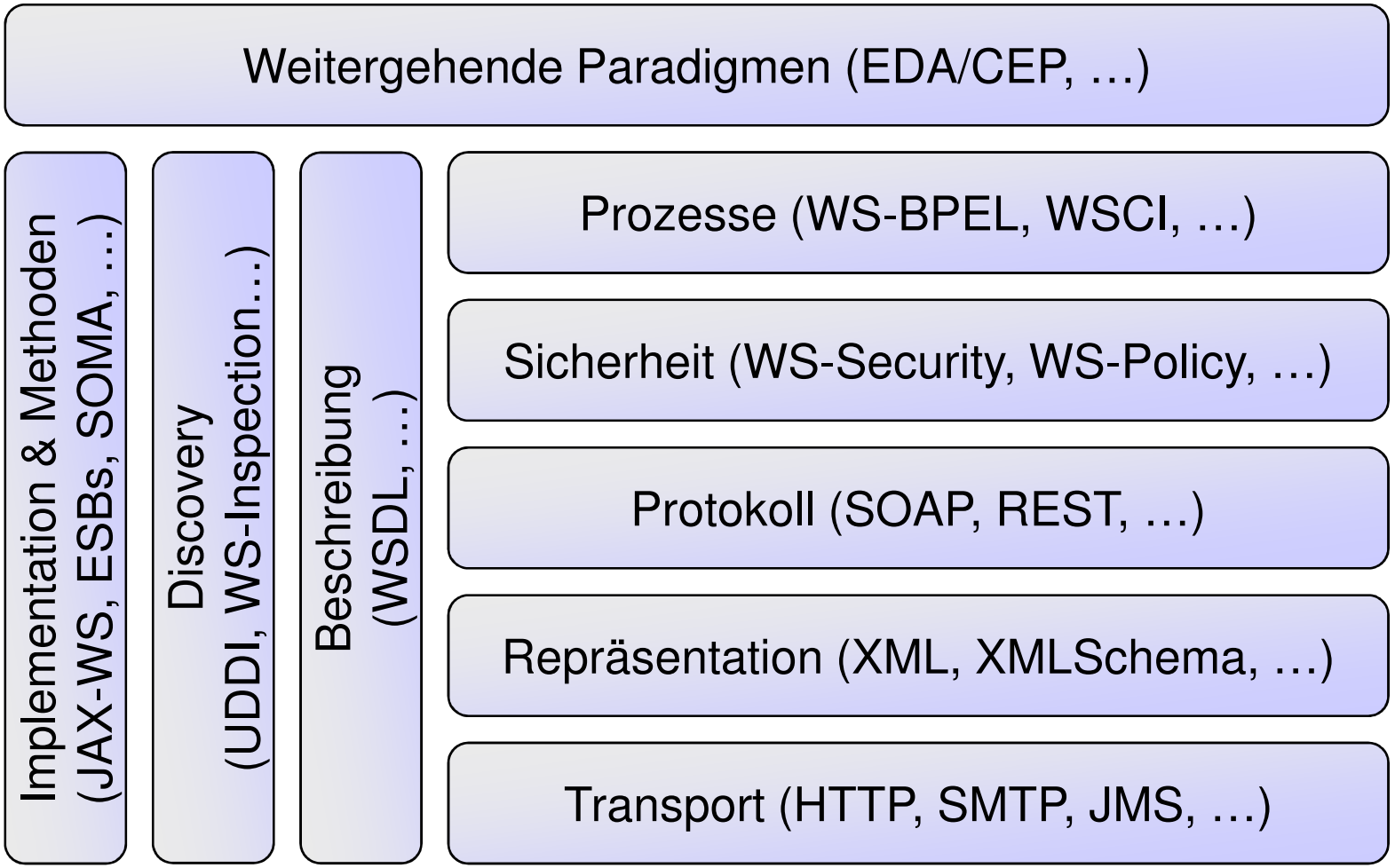
(Kandaswamy 2009)

ESB Produkte

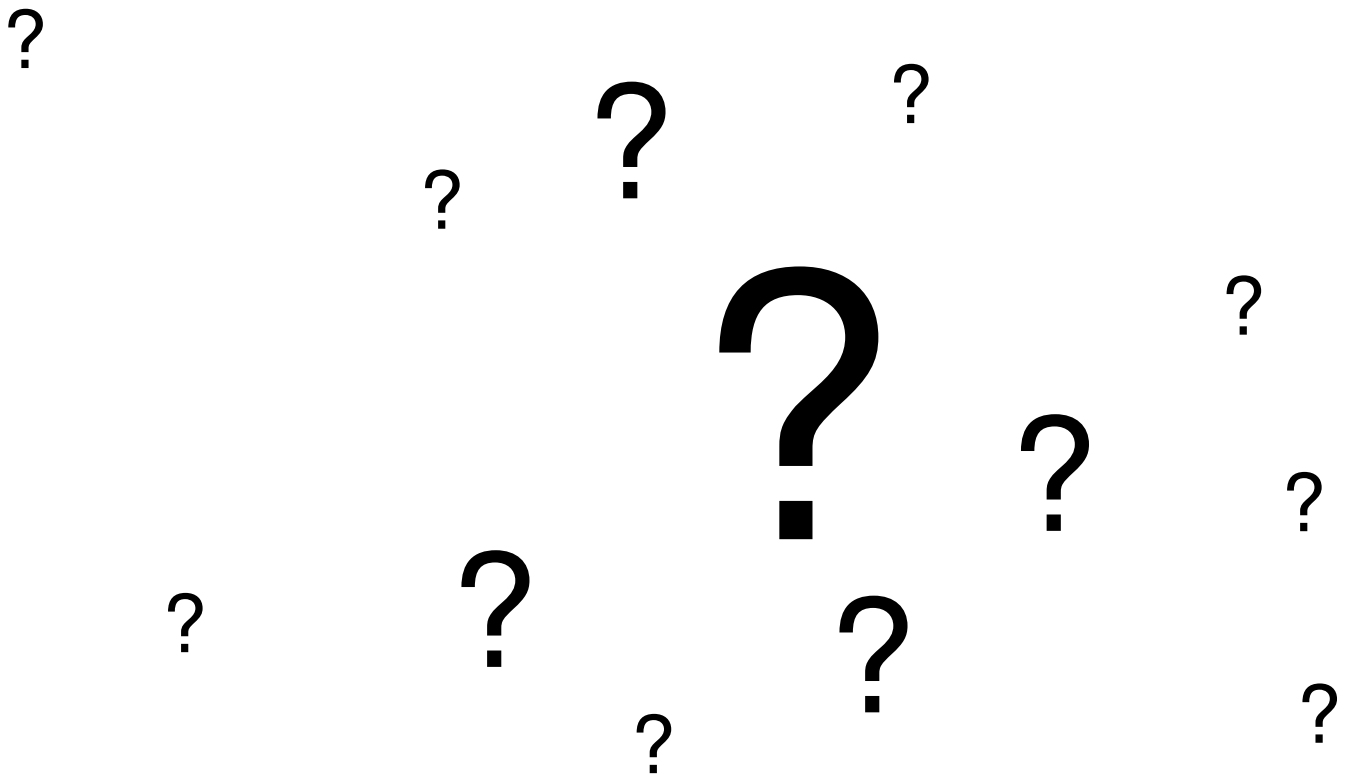
- Commercial
  - Adeptia ESB Suite
  - webmethods Enterprise Service Bus (SoftwareAG)
  - (TIBCO) ActiveMatrix™ BusinessWorks
  - IBM WebSphere ESB
  - IBM WebSphere Message Broker
  - Microsoft BizTalk Server
  - Neudesic Neuron ESB
  - Windows Azure Service Bus
  - Oracle Enterprise Service Bus (BEA Logic)
  - Progress Sonic ESB (acquired by Trilogy)
  - Red Hat JBoss Fuse
  - IONA (acquired by Progress)
  - InterSystems Ensemble
- Open Source
  - Apache ServiceMix
  - Apache Synapse
  - JBoss ESB
  - NetKernel
  - Petals ESB
  - Spring Integration
  - Open ESB
  - WSO2 ESB
  - Mule
  - UltraESB
  - Red Hat Fuse ESB (based on Apache ServiceMix)
  - Talend
  - Zato. ESB and application server. Open-source. In Python.

(Wikipedia 2014)

Zusammenfassung und Ausblick



Fragen



Web Services: Entwicklung bis 2004

Weite Unterstützung in der Industrie **(HP, IBM, Microsoft, Bea, Ariba…)**

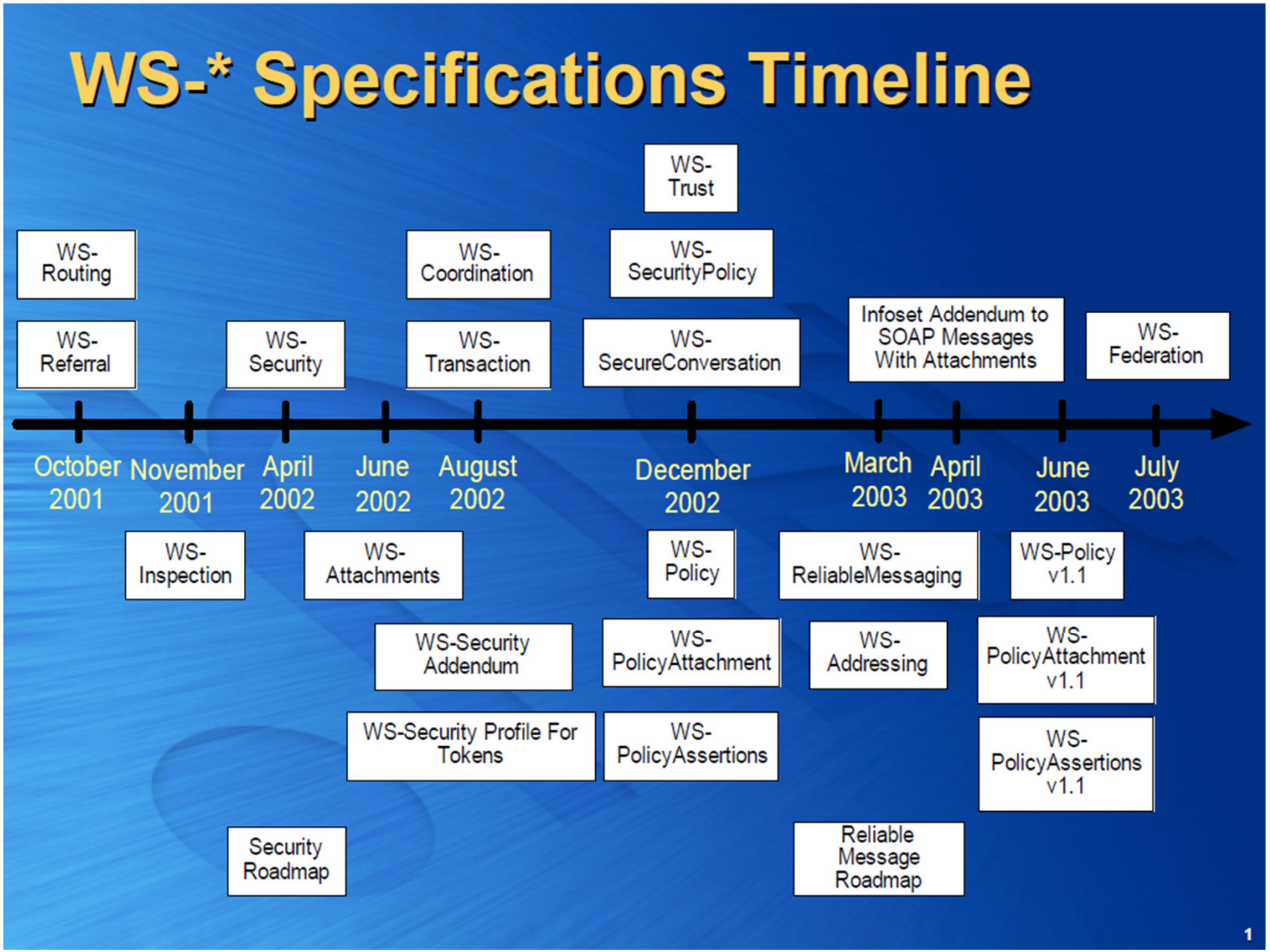
2004 bereits über 20 verschiedene „WS-xx-Specifications“ verabschiedet (!)

Aber: das „Rad“ mehrfach „neu erfunden“ !

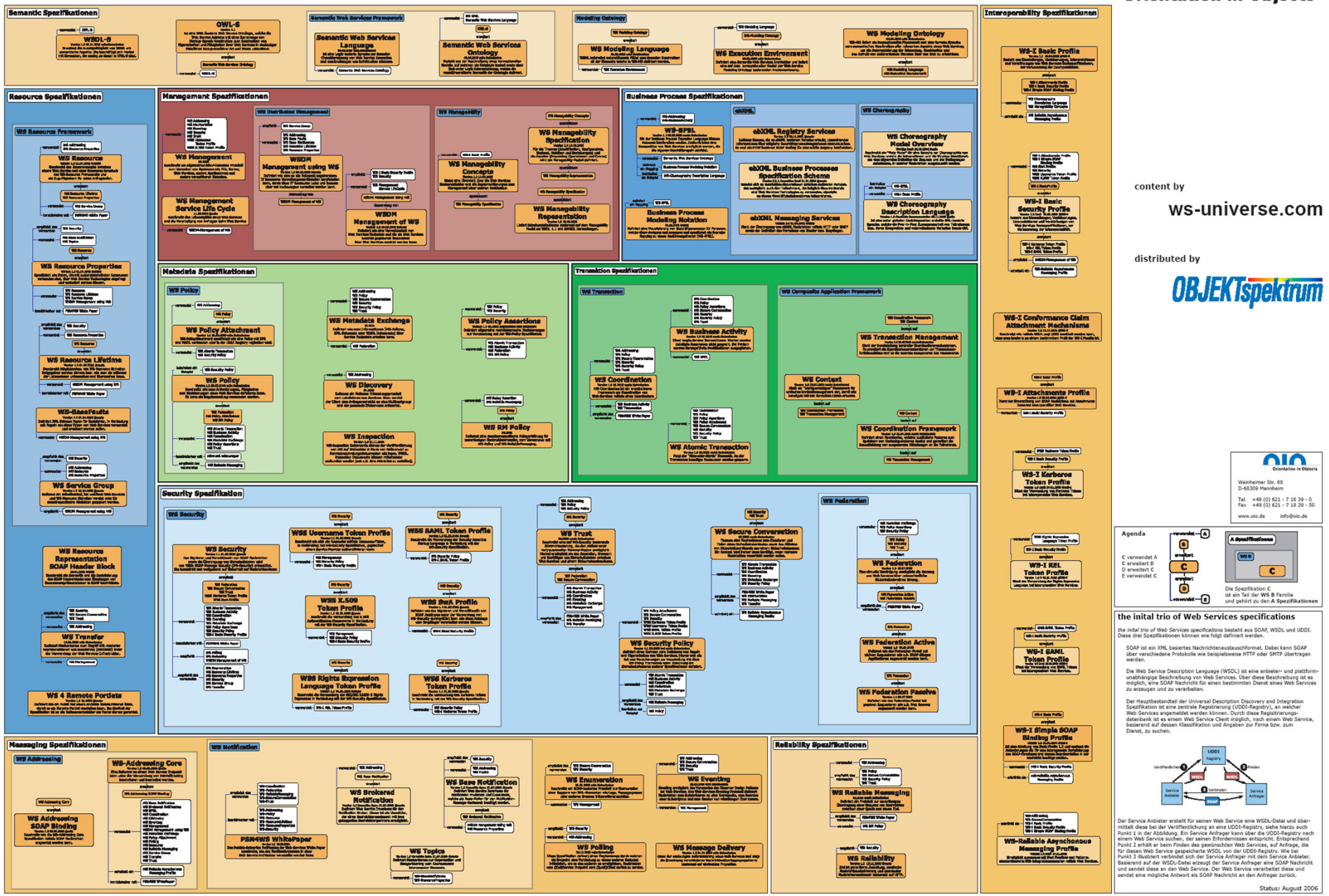
Offene Fragen (u.a.):

- Interoperabilität (??)
- Sicherheit (inzwischen teilweise adressiert)
- Vertrauen in Registry-Betreiber (!)
- Bezug zu neueren Anwendungen wie z.B. „Mobilität“
- Akzeptanz (?)
- etc.

.. und August 2006


WS-* Specifications Timeline
WS-Trust
WS-Routing
WS-Coordination
WS-SecurityPolicy
WS-Referral
WS-Security
WS-Transaction
WS-SecureConversation
Infoset Addendum to SOAP Messages With Attachments
WS-Federation
October 2001
November 2001
April 2002
June 2002
August 2002
December 2002
March 2003
April 2003
June 2003
July 2003
WS-Inspection
WS-Attachments
WS-Policy
WS-ReliableMessaging
WS-Policy v1.1
WS-Security Addendum
WS-PolicyAttachment
WS-Addressing
WS-PolicyAttachment v1.1
WS-Security Profile For Tokens
WS-PolicyAssertions
WS-PolicyAssertions v1.1
Security Roadmap
Reliable Message Roadmap
1

Spezifikationen im Web Service Umfeld



Orientation in Objects

content by

ws-universe.com

distributed by

OBJEKTspektrum

Orientation in Objects
Weinheimer Str. 68
D-68309 Mannheim
Tel. +49 (0) 621 - 7 18 39 - 0
Fax +49 (0) 621 - 7 18 39 - 50
www.oio.de info@oio.de

Agenda

C verwendet A
C erweitert B
D erweitert C
E verwendet C

Die Spezifikation C ist ein Teil der **WS B** Familie und gehört zu den **A Spezifikationen**

the inital trio of Web Services specifications

the inital trio of Web Services specifications besteht aus SOAP, WSDL und UDDI. Diese drei Spezifikationen können wie folgt definiert werden.

SOAP ist ein XML basiertes Nachrichtenaustauschformat. Dabei kann SOAP über verschiedene Protokolle wie beispielsweise HTTP oder SMTP übertragen werden.

Die Web Service Description Language (WSDL) ist eine anbieter- und plattformunabhängige Beschreibung von Web Services. Über diese Beschreibung ist es möglich, eine SOAP Nachricht für einen bestimmten Dienst eines Web Services zu erzeugen und zu verarbeiten.

Der Hauptbestandteil der Universal Description Discovery and Integration Spezifikation ist eine zentrale Registrierung (UDDI-Registry), an welcher Web Services angemeldet werden können. Durch diese Registrierungsdatenbank ist es einem Web Service Client möglich, nach einem Web Service, basierend auf dessen Klassifikation und Angaben zur Firma bzw. zum Dienst, zu suchen.

Der Service Anbieter erstellt für seinen Web Service eine WSDL-Datei und übermittelt diese bei der Veröffentlichung an eine UDDI-Registry, siehe hierzu auch Punkt 1 in der Abbildung. Ein Service Anfrager kann über die UDDI-Registry nach einem Web Service suchen, der seinen Erfordernissen entspricht. Entsprechend Punkt 2 erhält er beim Finden des gewünschten Web Services, auf Anfrage, die für diesen Web Service gespeicherte WSDL von der UDDI-Registry. Wie bei Punkt 3 illustriert verbindet sich der Service Anfrager mit dem Service Anbieter. Basierend auf der WSDL-Datei erzeugt der Service Anfrager eine SOAP Nachricht und sendet diese an den Web Service. Der Web Service verarbeitet diese und sendet eine mögliche Antwort als SOAP Nachricht an den Anfrager zurück.

Status: August 2006

Mythen und Legenden zu Web Services

- Web Services sind einfach
- Web Services benötigen keine Programmierung
- Web Services sind nicht sicher
- Web Services sind per definitionen interoperabel
- Web Services sind an HTTP gebunden
- Web Services sind synchrone RPC-Aufrufe
- Web Services sind Punkt-zu-Punkt-Verbindungen
- Web Services sind langsam

(Melzer et al. 2007)

Web Service-Protokollebenen

Discovery	UDDI
Description	WSDL
Messaging	SOAP, XML-RPC
Transport	HTTP, SMTP, FTP

Transport: Verantwortlich für den Nachrichtentransport zwischen den Anwendungen

Messaging: Verantwortlich für Transport und Kodierung der Nachrichten (in XML)

Description: Verantwortlich für Beschreibung der öffentlichen Schnittstelle des Dienstes (*funktional, prozedural, quality-of-service*)

Discovery: Verantwortlich für das Zusammenfassen der Dienste in einer allgemeinen „Registrierung“ und deren leichtes Einstellen und Auffinden.

SOAP: Beispiel

- **SOAP Anfrage**

```
Code
1  <env Envelope
2  xmlns:env=''http://www.w3.org/2003/05/soap-envelope''>
3  <env:Body>
4  <GetServerTime></GetServerTime>
5  </env:Body>
6  </env:Envelope>
```

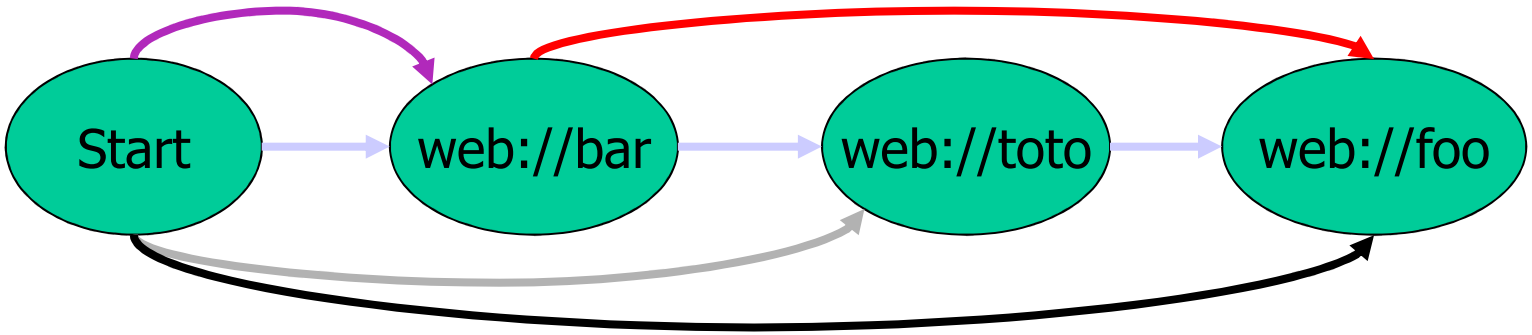
- **SOAP Antwort**

```
Code
1  <env Envelope
2  xmlns:env=''http://www.w3.org/2003/05/soap-envelope''>
3  <env:Body>
4  <GetServerTimeResponse>
5  <Now>04/01/2007 12:00:00 AM</Now>
6  </GetServerTimeResponse>
7  </env:Body>
8  </env:Envelope>
```

SOAP Header

ermöglicht modulares Hinzufügen von Eigenschaften

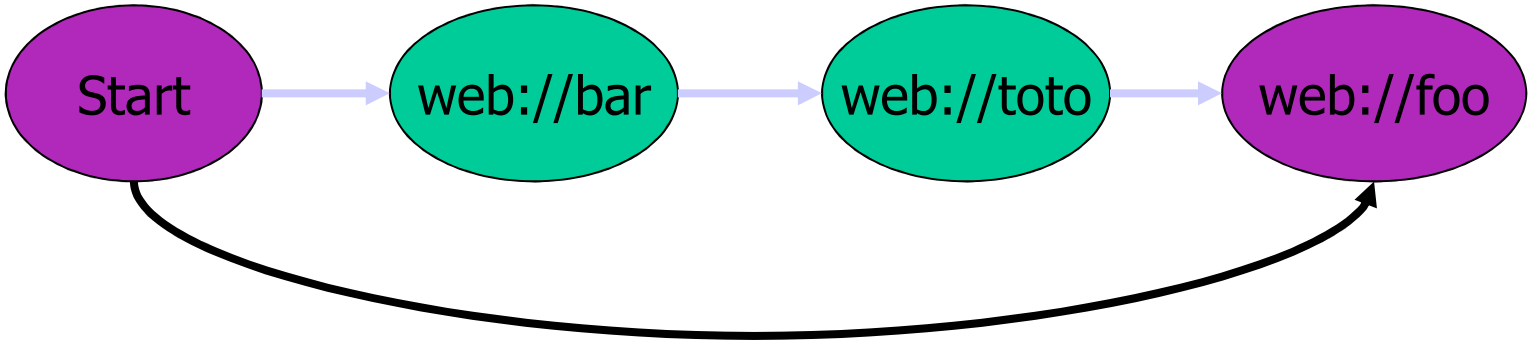
- beliebige Anzahl (auch kein Header möglich)
  - Authentifizierung, Verschlüsselung etc.
- Adressierung von SOAP-Prozessoren über „role“-Attribut
  - „next“, „ultimateReceiver“, “none“ oder eigene „role identifier“
- optional/verpflichtend über „mustUnderstand“-Attribut
  - definierte Reaktion auf Verarbeitungsfehler (graceful degradation)



SOAP Body

Nachrichteninhalt

- kann nicht weggelassen werden
- Verarbeitungsmodell entspricht Header mit
  - mustUnderstand = "true"
  - role = "ultimateReceiver“
- enthält (ein) beliebiges XML



SOAP Fault Element

Fehlerbehandlung expliziter Teil des SOAP Verarbeitungsmodells

- vgl. exceptions, try/catch…
- beschreibt nur Verarbeitungsfehler (keine Kommunikationsfehler)
- → explizit geworfene Exceptions
- *Fault Element* im SOAP Body

Fault Element enthält

- Fehlercode, Fehlergrund als (menschenlesbarer) Text
- optional Knoten oder Rolle des Knotens, der den Fehler verursacht hat
- optionale (maschinenlesbare) Detailinformationen (anwendungsabhängig)

Mögliche Fehler

- vordefiniert (value): VersionMismatch, MustUnderstand, DataEncodingUnknown, Sender, Receiver
- Subcode (rekursiv): anwendungsabhängig

SOAP und Binärdaten

Beliebige Daten durch SOAP tunneln

- verschlüsselte Daten, Multimedia-Dateien
- SOAP Envelopes

Zwei Möglichkeiten

- direkt im Envelope (Body) als BLOB
- als Referenz (Übertragung zusätzlich zum Envelope)

Vielfältige Arten von Referenzen denkbar

- MIME multipart, URL auf externe Ressource, …
- erweiterte Möglichkeiten für Integritäts-Checks

Erweiterbarkeit durch SOAP

Über Header können *zusätzliche Features* eingesetzt werden:

- Authentication, Security, Transaction Management etc.

SOAP unterstützt zwei Arten der *Kombination von Features* (composability):

- *vertikal:* verschiedene Features in einer Nachricht
- *horizontal:* verschiedene Knoten bieten verschiedene Features

Vertikale Komponierbarkeit

Unabhängige Features können koexistieren

```
<SOAP:Envelope xmlns:SOAP="http://schemas.xmlsoap.org/soap/envelope”
  SOAP:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
   <SOAP:Header>
      <a:authentication …>…</a:authentication>
      <s:security …> … </s:security>
      <t:transactions …> … </t:transactions>
      <p:payment …> … </p:payment>
   </SOAP:Header>
   <SOAP:Body>
      <m:mybody> … </m:mybody>
   </SOAP:Body>
</SOAP:Envelope>
```

WSDL Types und Messages

Types und Messages werden global im Dokument definiert und wiederverwendet

Typen erlauben programmiersprachenunabhängige Datenrepräsentation

- XML-Basisdatentypen
- komplexe Strukturen durch eigene XML-Schemata

Nachrichtentypen beschreiben Kommunikation zwischen Dienst und Aufrufer

- eindeutiger Name (im Dokument)
- Operationen verweisen auf Nachrichtennamen

Nachrichteninhalte über *Parts* definiert

- jeder Part ist ein name/value-Paar
- Nachrichten können mehrere Parts besitzen
- Part hat einen Typ (s.o.)

portTypes und WSDL Operations

portTypes als abstrakte Dienstbeschreibungen

- Menge zusammengehöriger Operationen
- Operationen ähneln Methodensignaturen
- portTypes ähneln (Java) Interfaces

Operationen

- besitzen einen Namen sowie Nachrichten
- Nachrichten sind <input> oder <output>
- vier Interaktionsmuster
  - One way: <input>
  - Request-Response: <input>, <output>
  - Solicit-Response: <output>, <input>
  - Notification: <output>

WSDL Binding und Port

Abstrakte Beschreibung enthält keine technischen Informationen

- Wie werden Nachrichten übermittelt?
- Wo kann der Service aufgerufen werden?

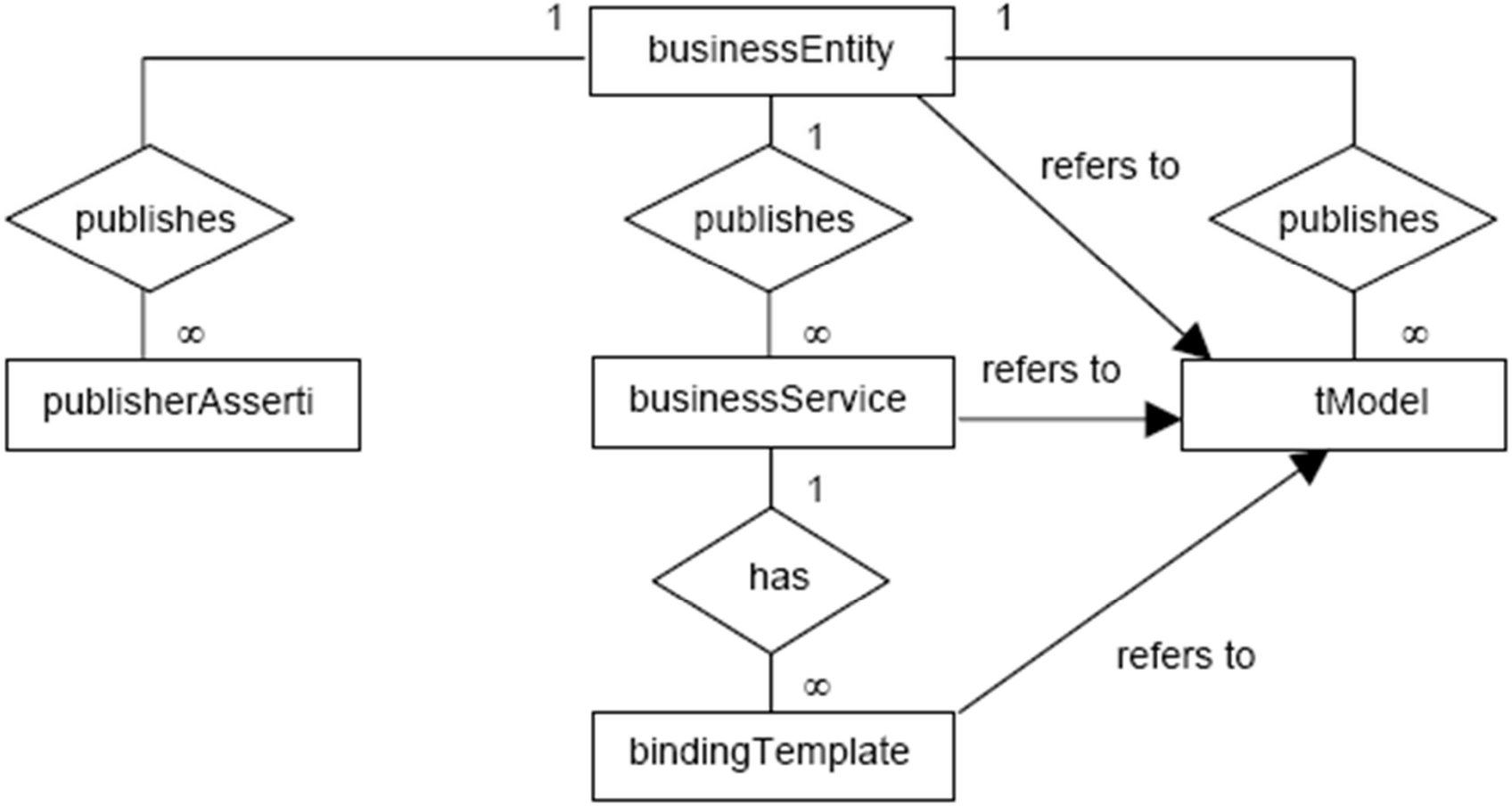
Bindings für portTypes

- verwendetes Protokoll (z.B. SOAP, HTTP GET/POST)
- protokollspezifische Informationen für Operationen (z.B. Repräsentation von Nachrichteninhalten im SOAP Body)

Adresse(n) eines Services

- Service kann mehrere Adressen enthalten
- spezifiziert für je ein Binding (z.B. soap:address)

UDDI Information Model



Information model of UDDI

Web Services: Einsatz und Vision

gestern:

- unternehmensinterner Einsatz, z.B. im Rahmen von EAI
- unternehmensübergreifender Einsatz, aufbauend auf bestehenden Geschäftsbeziehungen

heute:

- Abbildung komplexer Geschäftsprozesse
- Verwendung einfacher externer Dienste (z.B. Google APIs)

morgen?:

- (vollständig) automatisierte Dienstsuche und -nutzung
- Realisierung virtueller Unternehmen

WS Transport: HTTP

HyperText Transport Protocol (HTTP)

- einfaches request/response-Protokoll
- basiert auf TCP und ist somit verbindungsorientiert und zuverlässig
- für jedes Request/response-Paar wird eine eigene Verbindung aufgebaut: Es gibt keinen expliziten Zustand über mehrere Paare
- als Basisprotokoll des WWWs gelangt es leicht durch Firewalls

Nachrichtenübermittlung an den Server mit POST-Anfragen

```
POST /myFunctions/echo HTTP/1.1
Host: www.myserver.com
Content-Type:Text/plain
Content-Length: 12
Hello World!
```

Header-Daten

Anfrage-Daten

WS Messaging: SOAP

SOAP = XML-basiertes Transportprotokoll

- **SOAP** steht für SOAP (früher Simple Object Access Protocol)

SOAP-Spezifikation

- Nachrichtenformat
- Verarbeitungsmodell
- Bindung an darunter liegende Transportprotokolle (z.B. HTTP)
- Konvention für RPC-Aufrufsemantik
- Methoden um Binärdaten effizient zu verpacken

SOAP Envelope

Der **SOAP Envelope** ist das Wurzelelement einer SOAP-Nachricht

- grundlegende Einheit der Interaktion zwischen SOAP-Prozessoren

SOAP-Nachrichten sind Einweg-Übertragungen

- vom Sender über Vermittler zum Empfänger und
- können zu Nachrichtenmustern kombiniert werden (request/response)

Nachrichten werden entlang eines Pfades ge“routed“

- Verarbeitung in den Zwischenknoten möglich
- Jeder Knoten ist ein SOAP-Prozessor, der über URI identifizierbar ist

Alternativen zu SOAP

SOAP-Nachteile

- mächtiges aber kompliziertes Modell
- Transportunabhängigkeit führt zu Overhead

Alternative: REST = REpresentational State Transfer

- Idee: Architektur des Webs auf Webservices anwenden
- Jede URI ist eine eigene Ressource, wie im Browser-basierten Web
- URIs können ge-“bookmarked„ und ge-“cached“ werden
- HTTP-Protokoll als Basis

Erfolgreicher Einsatz in der Praxis

- Amazon, Google, Flickr, …

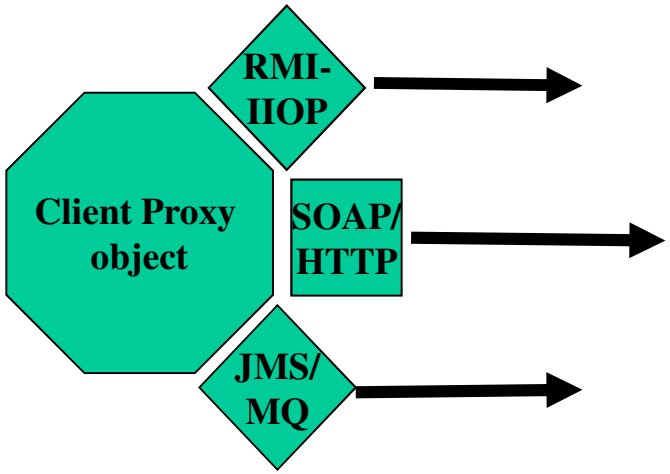
WSDL - Beispiel

```
1 <definitions targetNamespace="vsis/mwse" name="MobileWorkflowWebserviceService">
2   <types>
3     <xsd:import namespace="vsis/mwse" schemaLocation="http://vsisls4:30100/WorkflowServerEngine?xsd=1"/>
4   </types>
5   <message name="getWorkflow">
6     <part name="parameters" element="tns:getWorkflow"/>
7   </message>
8   <portType name="MobileWebservice">
9     <operation name="getWorkflow">
10      <input message="tns:getWorkflow"/>
11      <output message="tns:getWorkflowResponse"/>
12      <fault message="tns:ServerErrorException" name="ServerErrorException"/>
13    [...]
14  <binding name="MobileWebservicePortBinding">
15    <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document"/>
16  </binding>
17  <service name="MobileWorkflowWebserviceService">
18    <port binding="tns:MobileWebservicePortBinding">
19      <soap:address location="http://vsisls4:30100/WorkflowServerEngine"/>
20    </port>
21  [...]
```

Service-Aufruf (Idee)

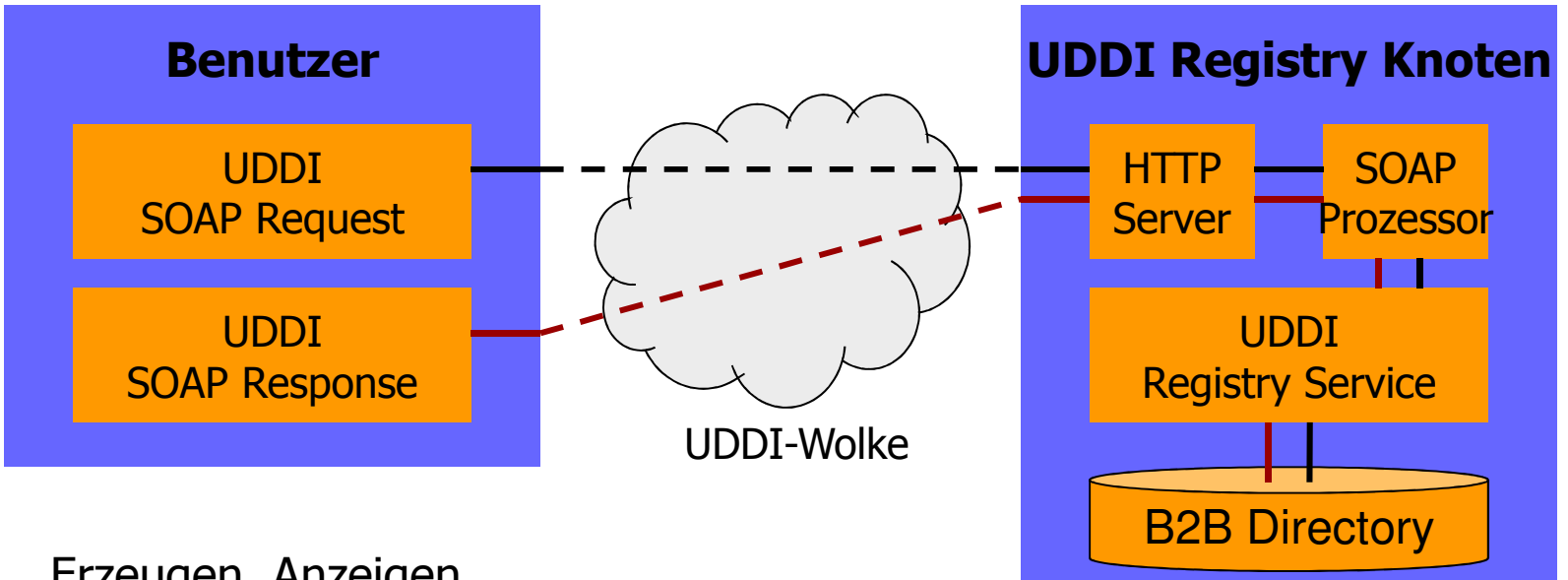
Aufruf erfolgt transparent über Binding

- aufrufender Code muss nur die abstrakte Schnittstelle kennen
- neue Bindings können dynamisch hinzugefügt werden
- ermöglicht Optimierung/Anpassung an Kontext



UDDI und SOAP

Die *UDDI Registry* ist über eine *SOAP-Schnittstelle* ansprechbar:



Erzeugen, Anzeigen, Aktualisieren und Löschen von registrierten Daten

implementierungsneutrale Umsetzung