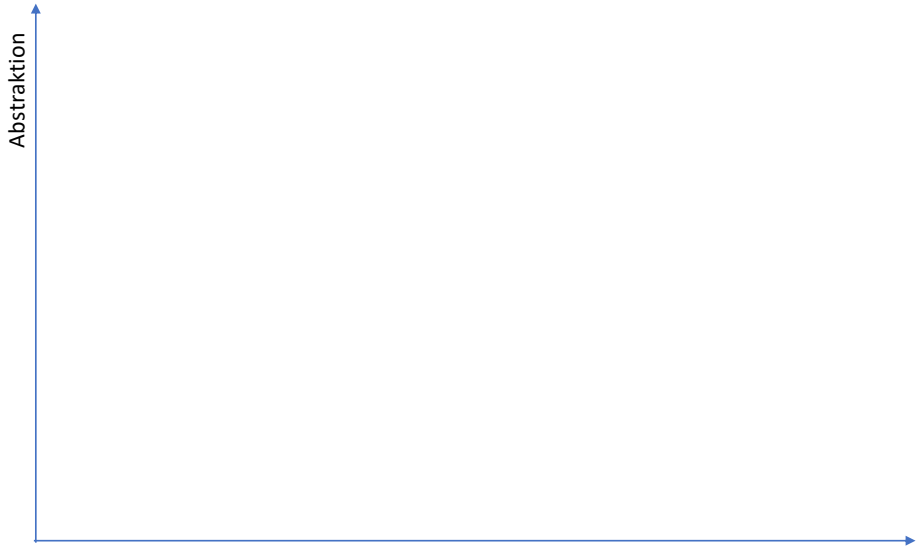
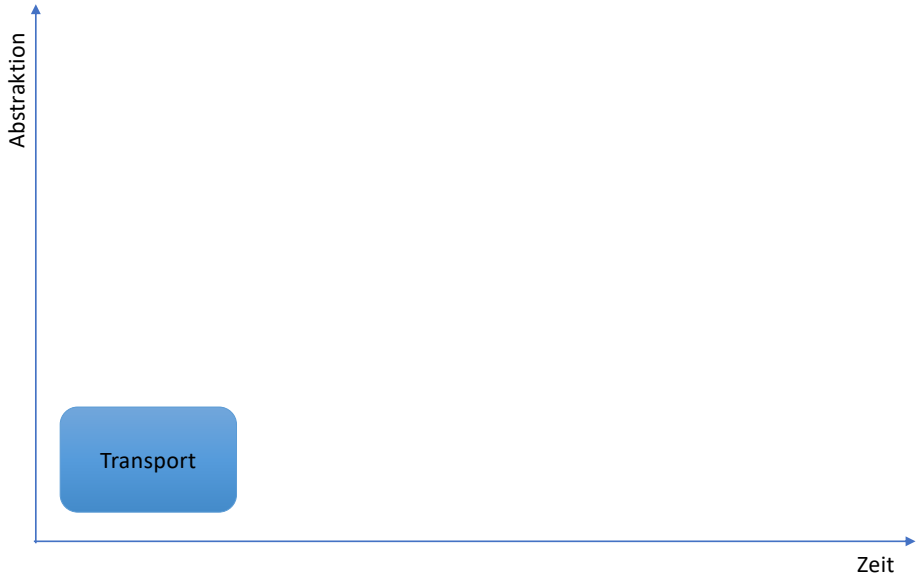

Software
Architectures
for Enterprises
SA4E
Peter Sturm
Universität Trier
Winter 2024


3. Deskriptiv
2

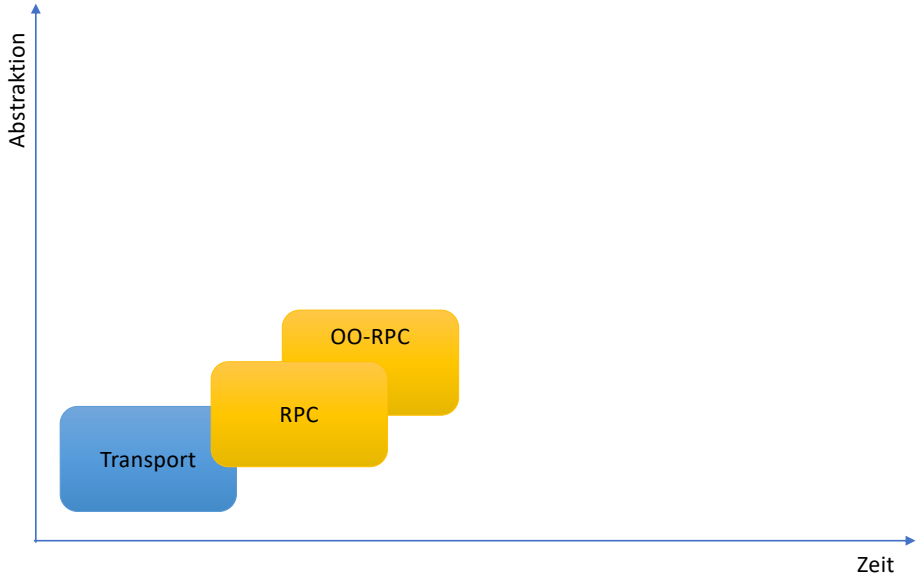
Es war einmal …



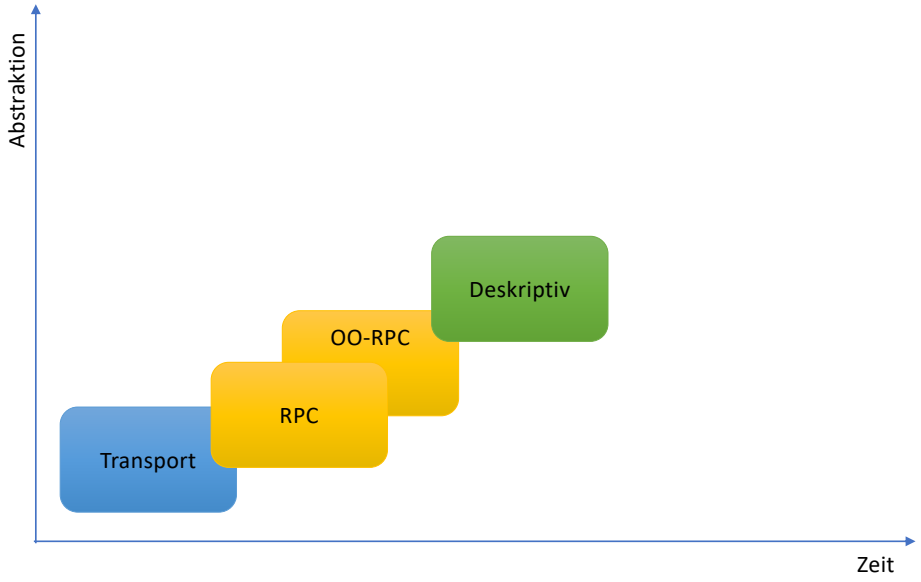
Eureka! Es funktioniert!

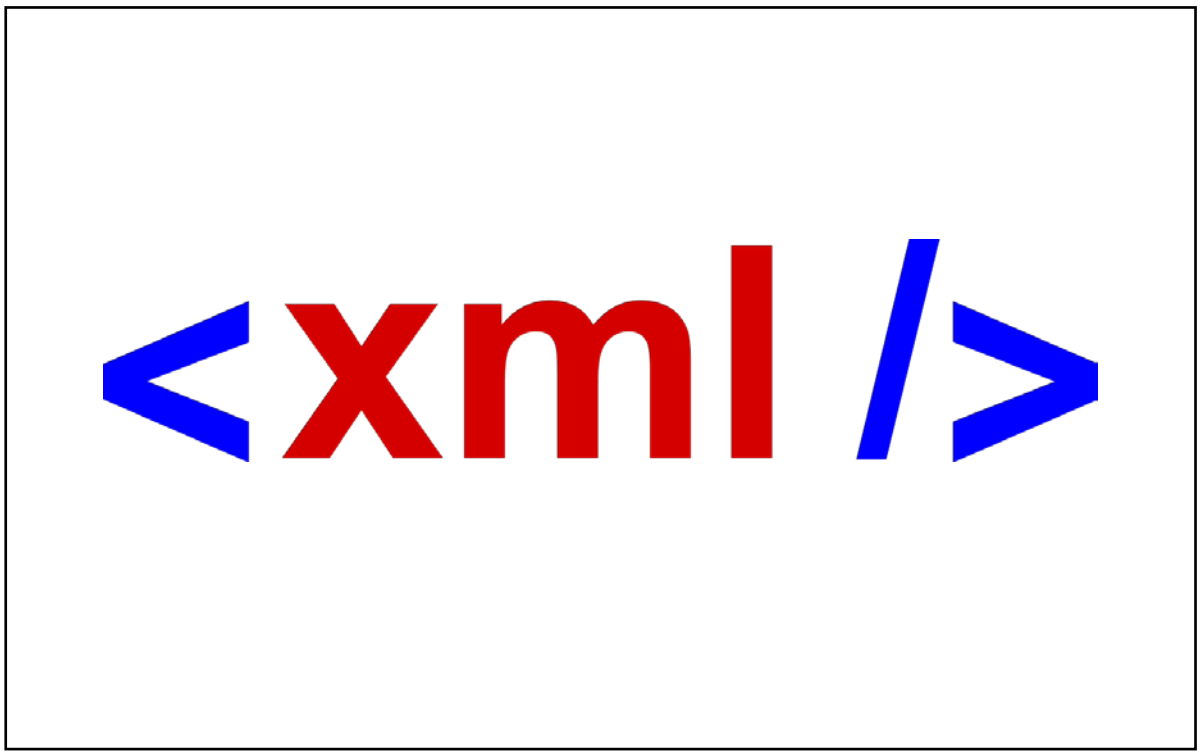


Client/Server



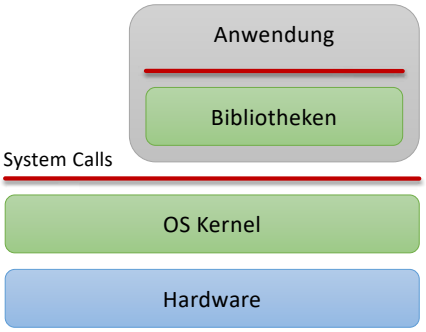
Noch höher!?




<xml />

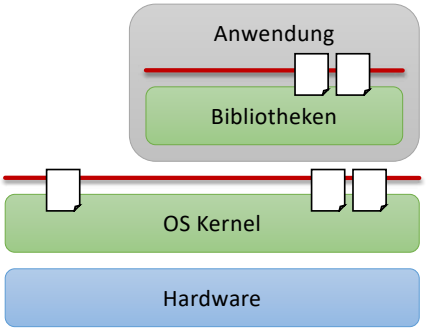
Bisher: 99% Programmieren

- Feste APIs
  - System-Call-Schnittstelle zum Kernel
  - Bibliotheksfunktionen
- Eventuell proprietäre Konfiguration
- Nativ oder interpretiert
- Nachteil
  - Starre Struktur
  - Eingeschränkte Parametrisierbarkeit
    - Argumente
    - Tunnelfunktionen (ioctl)
  - Programmierer

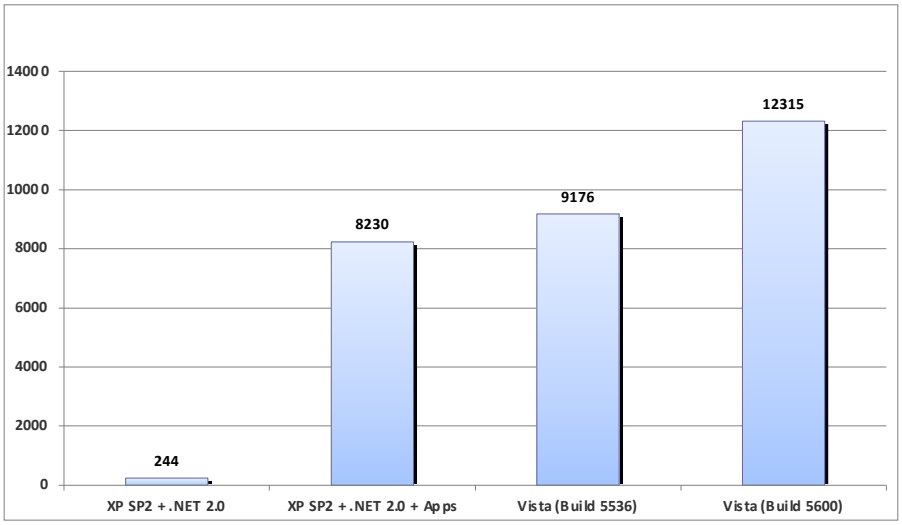


Zukünftig: 99% Beschreiben

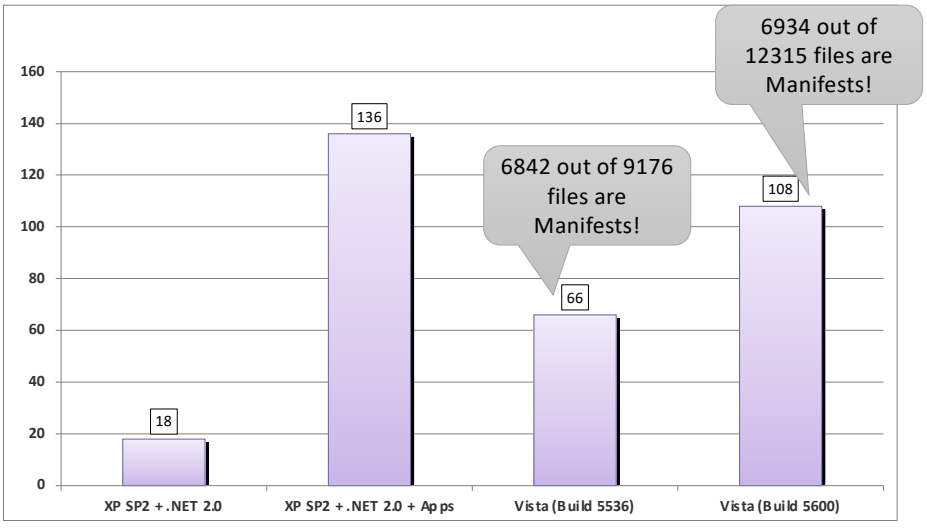
- Massives Weiten der APIs
  - XML-Dokumente beschreiben verschiedenste Anwendungs-, Laufzeit- und Systemeigenschaften
- XML wird DIE kommunizierte Syntax
- Vorteil
  - Vielfältige Einflußmöglichkeiten
  - Keine Programmierkenntnisse
  - Ein Standard ⇒ Synergie
- Nachteil
  - Überschaubarkeit gewährleisten
  - Höhere Anforderungen an Entwickler



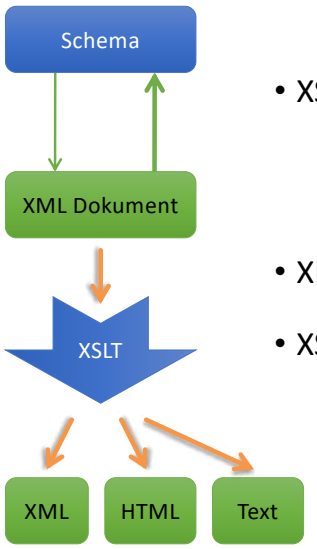
#XML Files



#Suffixes for XML Files

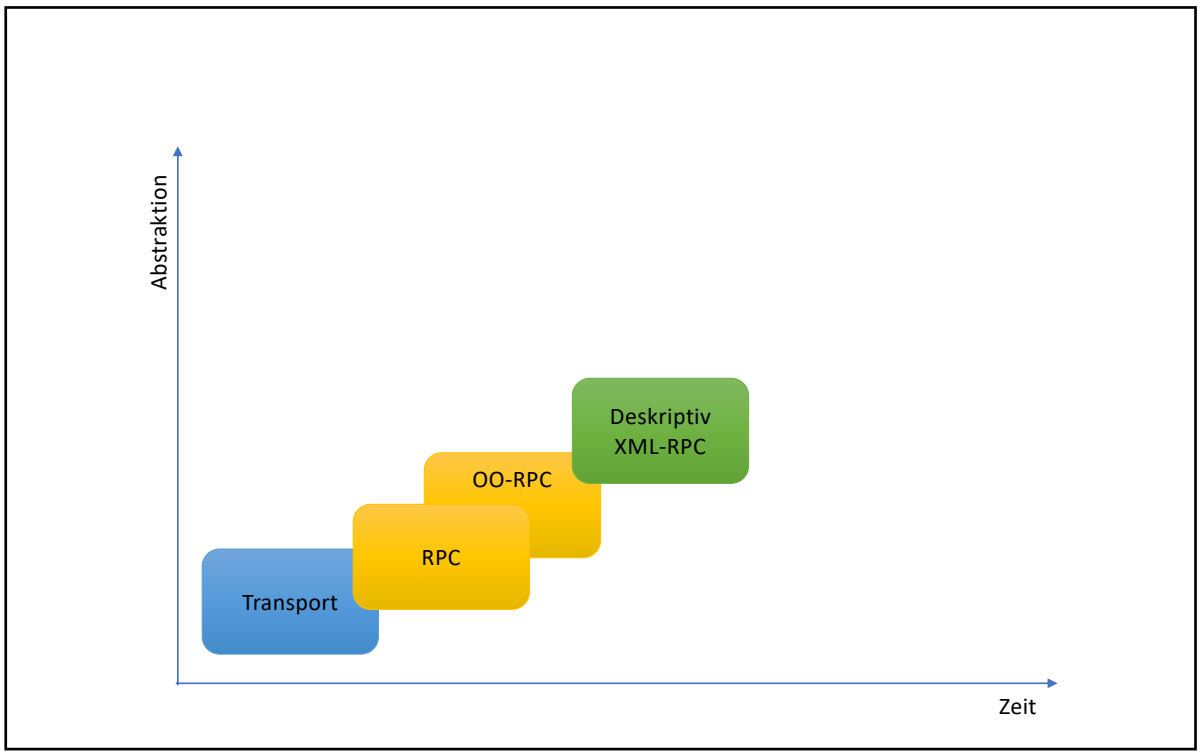


XML, XML Schema, XPath, XSL usw.



- XSD
  - Grammatik
  - Schema für ein oder mehrere Dokumente automatisiert erstellen
- XPath
- XSL (eXtensible Stylesheet Language)
  - XSLT und XSL-FO
  - Transformation bzw. Formatierung

RPC + XML?



Nachteile

- XML ist geschwätzig
  - Binärformate
- XML-Prozessoren/-Tools speicherhungrig
  - Lernprozeß?
- Mangelnde Zeiteffizienz?
  - Aufwendige Ansätze (DOM-Baum)
  - Lernprozeß?
  - MultiCores

Vorteile

- Synergieeffekte
  - XML = Lingua Franca
  - XML-Werkzeuge
  - Emergenz
- Lose Kopplung
  - Semantisch
  - Strukturell
- Mehr Automatisierung
- Dynamisch änderbare Strukturen
  - Service-orientierte Architekturen (SOA)
  - Servicebus

WebServices

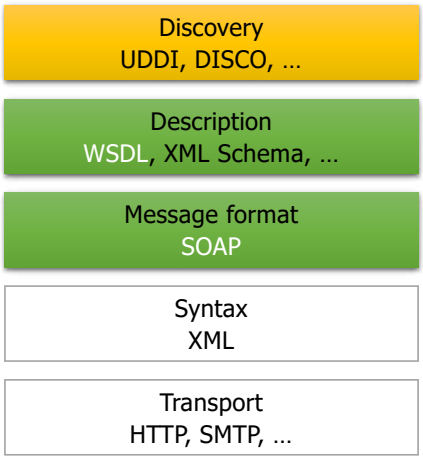
Starting Point

- Client/Server-based systems
  - Traditional RPC
  - RMI or Remoting
- Still often used
  - ONC RPC, COM+, Corba, Java RMI, .NET Remoting, ...
- Problems
  - Hard to manage
  - Legacy platforms
  - Limited or no interoperability
  - Leaving or entering secured networks complicated
  - Security and authentication issues

Web Services

- Deskriptive RPC
  - XML based $\Rightarrow$ Interoperability
- Reflective and virtual runtime plattforms
  - No IDL compiler and explicit stubs required
  - Again interoperability
- HTTP is used as a tunneling protocol

Pieces



- Discovery
  - Look for required service
  - Enhanced directory service
  - Semantic
  - Ontologies
- Transport
  - Most transport layer protocols usable
  - Concentration on HTTP and SMTP

Webservice

WS Client

Web Service

Webservice

WS Client

UDDI
Registry

Web Service

Webservice

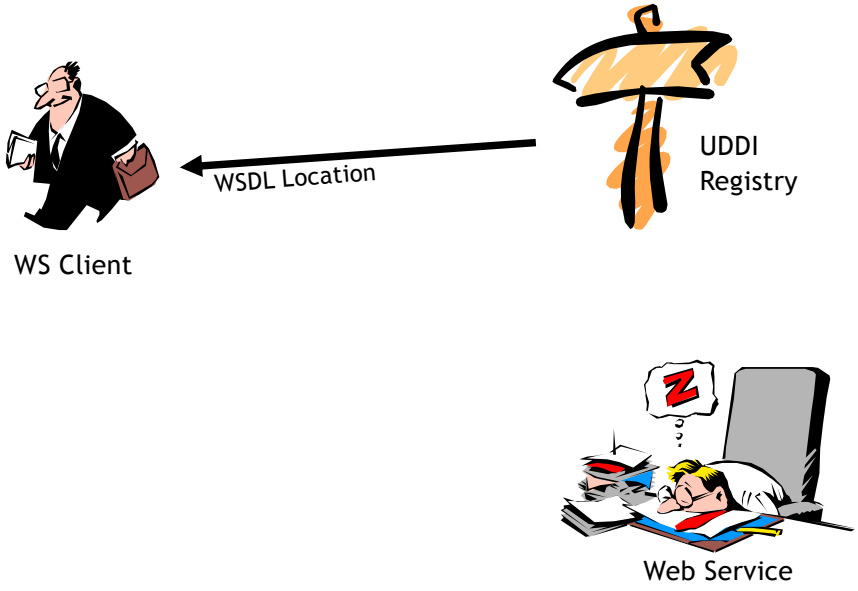
WS Client

Query Service

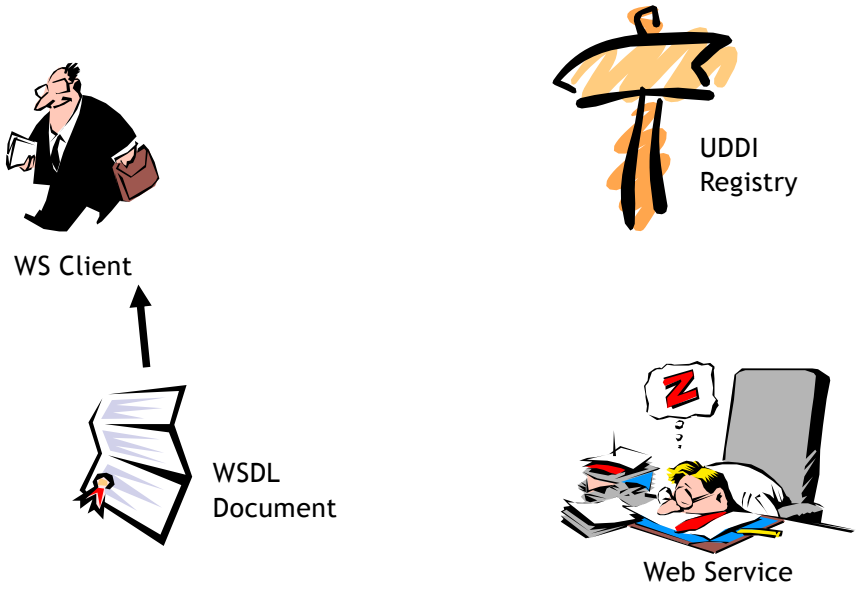
UDDI
Registry

Web Service

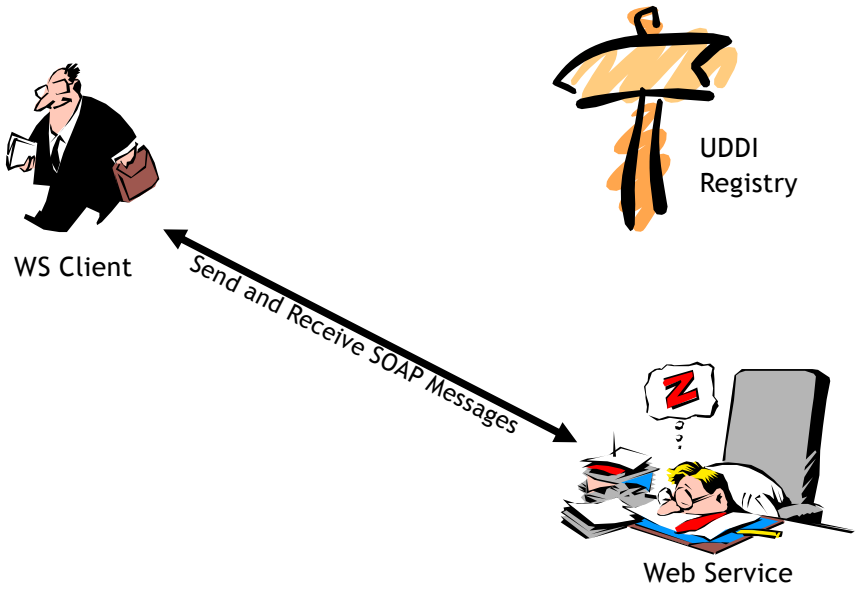
Webservice



Webservice

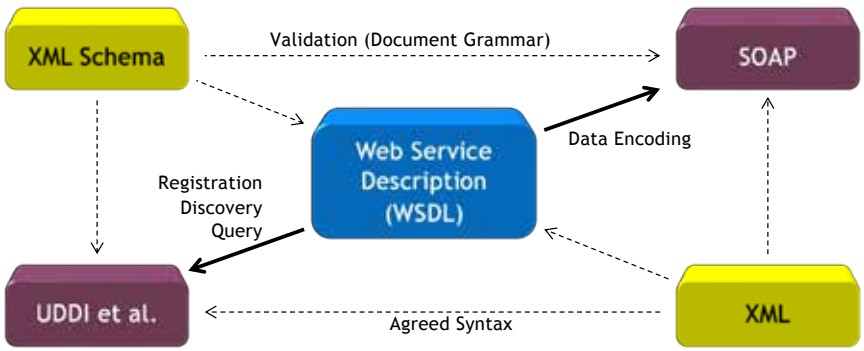


Webservice

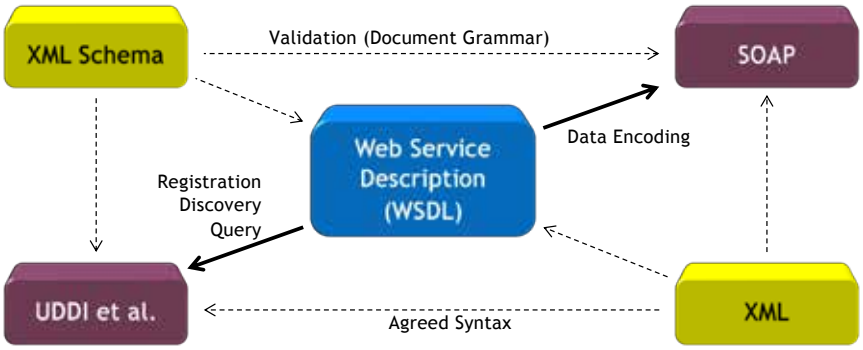


Webservices and XML

- Next Generation Client/Server
- Heavily based on XML



Webservices and XML



SOAP

Motivation

- SOAP = Simple Object Access Protocol
- Goals
  - XML schema is a grammar
  - Communicate true data structures
  - Must be interoperable
  - Describe complex dynamic structures like trees, …

SOAP Letters

- SOAP envelop = root element
  - May have a SOAP header
  - Must have a SOAP body
- Header
  - Augments message with meta information
- Body
  - Contains any data that can be encoded in XML

Example SOAP Request

```
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <soap:Body>
    <f1 xmlns="http://tempuri.org/">
      <v>42</v>
    </f1>
  </soap:Body>
</soap:Envelope>
```

Corresponding SOAP Response

```
HTTP/1.1 200 OK
Server: Microsoft-IIS/5.1
Date: Thu, 03 Nov 2005 05:47:51 GMT
X-Powered-By: ASP.NET
X-AspNet-Version: 1.1.4322
Cache-Control: private, max-age=0
Content-Type: text/xml; charset=utf-8
Content-Length: 322

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<soap:Body>
  <f1Response xmlns="http://tempuri.org/">
    <f1Result>43</f1Result>
  </f1Response>
</soap:Body>
</soap:Envelope>
```

SOAP Messaging Modes

- RPC/Literal
  - Element name = name of web service method requested
  - Arguments to method look like a struct
- RPC/Encoded (Prohibited in WS standard BP-1)
  - Based on XML Schema types
  - Data are represented as a graph of objects
- Document/Literal
  - Body is a well-formed XML element with separate namespace
- Document/Encoded (Prohibited in WS standard BP-1)

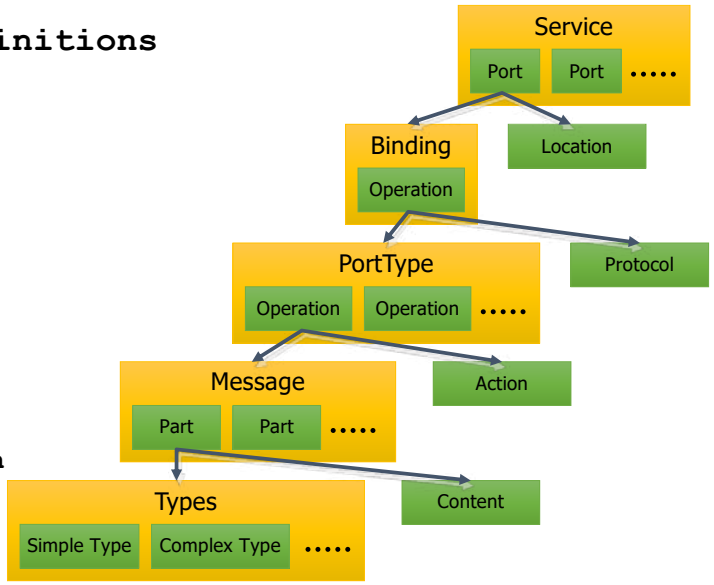
WSDL

WSDL

- Web Service Description Language
  - WSDL schema definition
  - XML document
- Goal: Describing a web service precisely

Basic Structure

- Root element **`definitions`**
- Nested elements
  - **`types`**
  - **`import`**
  - **`messages`**
  - **`portType`**
  - **`operations`**
  - **`binding`**
  - **`service`**
- Anywhere
  - **`documentation`**



import

- Include definitions from a specified namespace in another WSDL document
- Example
  - Modularize a complex WSDL document (separation of types, ports, etc.)
  - Maintain definitions for different services, but make a complete description accessible for clients
- Syntax
  - `<import namespace="URI" location="URI" />`

types

```
<wsdl:types>
  <s:schema elementFormDefault="qualified" targetNamespace="http://tempuri.org/">
    <s:element name="f1">
      <s:complexType>
        <s:sequence>
          <s:element minOccurs="1" maxOccurs="1" name="V" type="s:int"/>
        </s:sequence>
      </s:complexType>
    </s:element>
    <s:element name="f1Response">
      <s:complexType>
        <s:sequence>
          <s:element minOccurs="1" maxOccurs="1" name="f1Result" type="s:int"/>
        </s:sequence>
      </s:complexType>
    </s:element>
  </s:schema>
</wsdl:types>
```

- Defines all user-defined data structures required to communicate with the web service
- Basic type system = W3C XML schema built-in types

messages

```
<wsdl:message name="f1SoapIn">
  <wsdl:part name="parameters" element="tns:f1"/>
</wsdl:message>
<wsdl:message name="f1SoapOut">
  <wsdl:part name="parameters" element="tns:f1Response"/>
</wsdl:message>
```

- Defines message structure

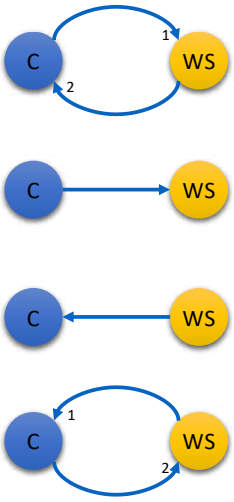
portType and operation

```
<wsdl:portType name="FunctionsSoap">
  <wsdl:operation name="f1">
    <wsdl:input message="tns:f1SoapIn"/>
    <wsdl:output message="tns:f1SoapOut"/>
  </wsdl:operation>
</wsdl:portType>
```

- **portType**
  - Defines an abstract interface to the service
  - One or more **operation** elements define the methods
- operation
  - **input** element and **output** element
  - **fault** element
- Operation overloading possible in WSDL
  - But not allowed in WS standard

Message Patterns

- Request/Response
  - **`input`** element follwed by **`output`** element
  - Additional **`fault`** elements may be included
- One-Way
  - Operation declared with single **`input`** element
  - No **`output`** and **`faults`** allowed
- Notification (not in standard)
  - **`output`** element only in operation
  - Push model for distributed systems
- Solicit/Response (not in standard)
  - Web service initiated Request/Response
  - **`output`** element followed by **`input`** element



`binding`

```
<wsdl:binding name="FunctionsSoap" type="tns:FunctionsSoap">
  <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
  <wsdl:operation name="f1">
    <soap:operation soapAction="http://tempuri.org/f1" style="document"/>
    <wsdl:input>
      <soap:body use="literal"/>
    </wsdl:input>
    <wsdl:output>
      <soap:body use="literal"/>
    </wsdl:output>
  </wsdl:operation>
</wsdl:binding>
```

- Maps the abstract interface **`portType`** to concrete protocols
  - SOAP, MIME, HTTP
- Defines the messaging style (RPC or document)
- Defines the encoding style (Literal or SOAP encoding)

`service` and `port`

```
<wsdl:service name="Functions">
  <documentation xmlns="http://schemas.xmlsoap.org/wsdl/"/>
  <wsdl:port name="FunctionsSoap" binding="tns:FunctionsSoap">
    <soap:address location="http://localhost/SOAPTest/Functions.asmx"/>
  </wsdl:port>
</wsdl:service>
```

- **`ports`**
  - Define the endpoint of a specific web service
  - Links to some binding
  - Variants
    - Different ports for different bindings
    - Multiple ports for the same binding
- **`service`**
  - Comprises a set of ports

Complete Document

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <wsdl:definitions
3     xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
4     xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
5     xmlns:s="http://www.w3.org/2001/XMLSchema"
6     xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
7     xmlns:tns="http://tempuri.org/"
8     xmlns:tm="http://microsoft.com/wsdl/mime/textMatching/"
9     xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
10    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
11    targetNamespace="http://tempuri.org/">
12    <wsdl:types>
30    <wsdl:message name="f1SoapIn">
33    <wsdl:message name="f1SoapOut">
36    <wsdl:portType name="FunctionsSoap">
42    <wsdl:binding name="FunctionsSoap" type="tns:FunctionsSoap">
54    <wsdl:service name="Functions">
55       <documentation xmlns="http://schemas.xmlsoap.org/wsdl/"/>
56       <wsdl:port name="FunctionsSoap" binding="tns:FunctionsSoap">
57          <soap:address location="http://localhost/SOAPTest/Functions.asmx"/>
58       </wsdl:port>
59    </wsdl:service>
60 </wsdl:definitions>
```

Remarks

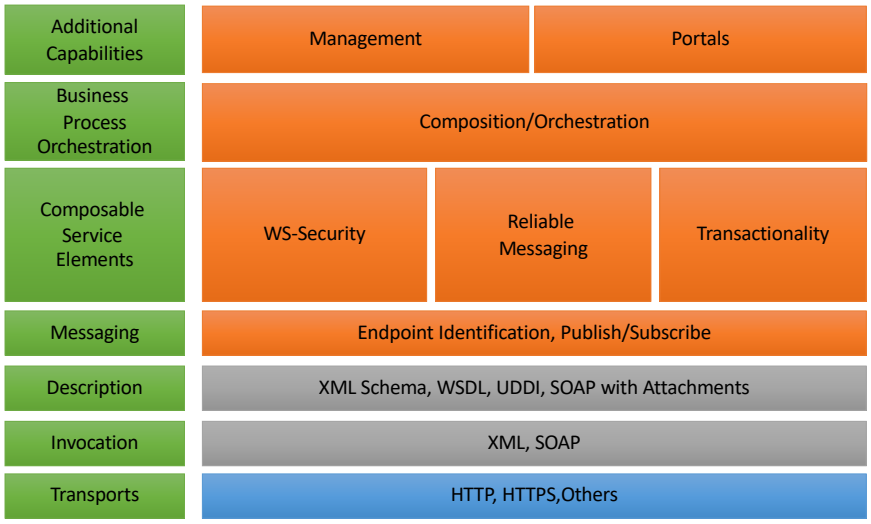
- WSDL descriptions can be quite “impressive”
- WSDL not meant to be used by humans
  - Development environments and runtime support
  - Generate the WSDL document for a new web service
  - Generate proxy code when importing a WSDL document

WS-I

Der Standard

- Web Services Interoperability Organization (WS-I)
  - http://ws-i.org
- Profil
  - Verschiedene Basistechniken und Protokolle
- Minimieren von “Reibungsverlusten”
- Weit verbreitet war “Basic Profile 1.0 (BP)”
  - SOAP 1.1, WSDL 1.1, UDDI 2.0
  - XML 1.0, XML Schema
  - HTTP 1.1

The Web Services Standards Stack



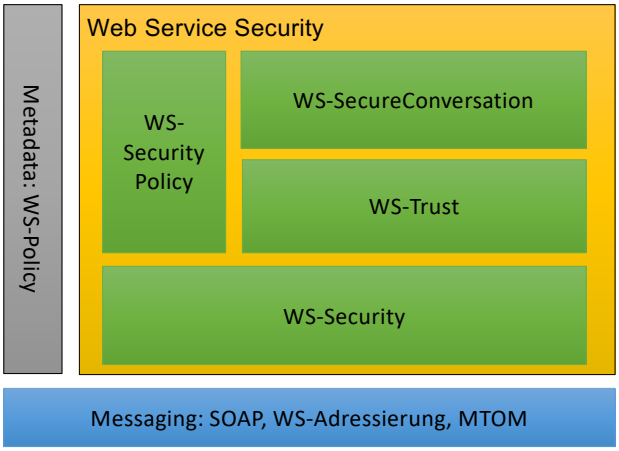
Basic Profile 1.0

- SOAP 1.1, WSDL 1.1, UDDI 2.0, XML 1.0, XML Schema und HTTP 1.1
- Mehr als 200 Interoperabilitätsprobleme gelöst
- Konventionen bzgl. Messaging, Description, und Discovery
  - Deprecation of RPC-encoded (stattdessen XML Schema)
  - Richtlinien bei der Nutzung von RPC/lit
  - Eindeutige Signaturen für Nachrichten
  - Fehlerbehandlung

BP 1.2 und BP 2.0

- Basic Profile
  - MTOM
    - Nachrichtenoptimierung (u.a. Binary XML)
  - WS-Addressing
    - Verallgemeinerung
  - SwA (SOAP with Attachments)
    - MIME Packaging
- Security Profile
  - Transport Layer Security (TLS, SSL, …)
  - Security Tokens (Username, X.509, REL, Kerberos, SAML, …)
  - Message and Attachment Security

Sicherheitsebenen



- WS-Security
  - Sicherheit auf SOAP-Ebene
- WS-Trust
  - Sicherheit im Umgang mit Tokens
- WS-SecureConversation
  - Sichere Sessions
- WS-SecurityPolicy
  - Sicherheitsanforderungen

WS-Security

- OASIS-Standard
- Grundlage für Web Services Security
  - Sicherheit auf SOAP-Nachrichtenebene
  - Message integrity und confidentiality
  - Einführung von Security Tokens
- Besteht aus mehreren Spezifikationen
  - Core
  - Token Profiles
    - Nutzung gängiger Credentials:
      - UserName/Password, X.509 Certificates, Kerberos, SAML

WS-Trust

- Standard der OASIS WS-SX TC
- Mechanismen für wechselseitiges Vertrauen
  - Security Token Service (STS)
    - Zuständig für Ausgabe, Validierung, Rücknahme und Erneuerung der Tokens
- Framework ist unabhängig von Protokoll und Tokentyp
- Definiert Nachrichtenformate zwischen Client und STS
  - Request Security Token (RST)
  - Request Security Token Response (RSTR)
  - Verhandlungen und Challenges
- Definiert “common patterns”
  - Security token issuance, renewal, cancellation, validation

WS-SecureConversation

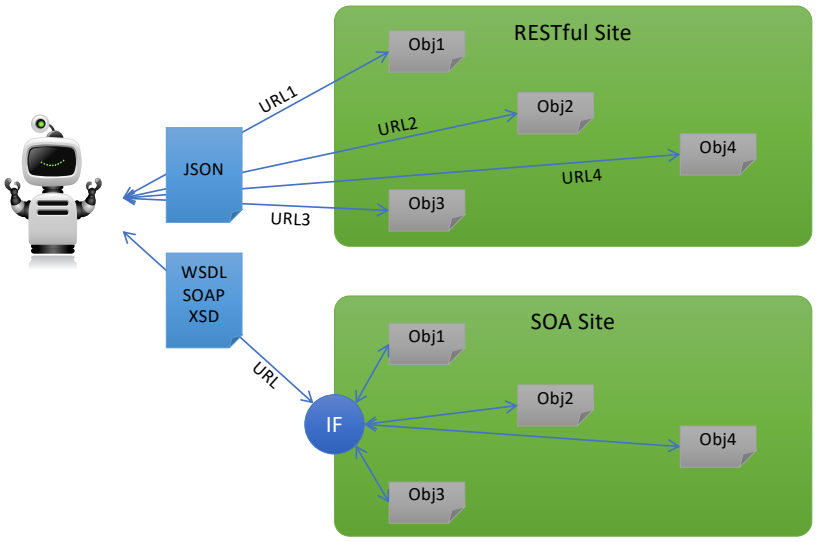
- Ebenfalls Standard von OASIS WS-SX TC
- Shared security context/session
  - Context enthält keys/secrets, Claims, u.a.
- Context wird über WS-Trust aufgebaut und unterhalten
- Hybride Struktur (vgl. PKI)
  - Aufwendige Authentifizierung nur einmal zu Beginn
  - Ergebnis wird auf beiden Seiten gespeichert: SecurityContext
  - Nur Context-Identifier wird in Nachrichten übertragen
    - Reduziert Nachrichtengröße
  - Nur symmetrische Verschlüsselung nach dem Context-Aufbau

WS-SecurityPolicy

- Standard der OASIS WS-SX TC
- Bestandteil einer WSDL-Beschreibung als WS-PolicyAttachment
- Legt Sicherheitsanforderungen fest
  - Was ist zu schützen?
  - Welche Tokentypen sind zu verwenden?
  - Algorithmen
  - Protokollversionen
  - …

RESTful

REST vs. WebServices



Historisches

- Dissertation Roy Fielding (2000)
- Architectural Styles and the
  Design of Network-based Software Archiectures
- Allgegenwärtiges WWW
- Nutzung in der Anwendung

„Vision“

- Nutzung in der Anwendung
- Ressourcen
- Zustandlose Kommunikation
- Hypermedia
- Repräsentation
- Standardisierte Zugriffsmethoden

REST

- REST = Representational State Transfer
- Verwendet HTTP-Operationen
  - GET (Anfrage)
  - HEAD (Anfrage, Metadaten)
  - POST (Create)
  - PUT (Update)
  - DELETE (Löschen)
- Jedes „Objekt“ hat eigene URI

Ressourcen

- Eindeutige Identifikation => URI

  `scheme://authority]path[?query][#fragment]`

- Beispiele

Typ	Beispiel
Primärressource	http://somewhere:4242/items/1
Subressource	http://somewhere:4242/items/1/price/22
Listenressource	http://somewhere:4242/items
Gefilterte Ressource	http://somewhere:4242/items?name=hmpf

JSON

```
{"menu": {
  "id": "file",
  "value": "File",
  "popup": {
    "menuitem": [
      {"value": "New", "onclick": "CreateNewDoc()"},
      {"value": "Open", "onclick": "OpenDoc()"},
      {"value": "Close", "onclick": "CloseDoc()"}
    ]
  }
}}
```

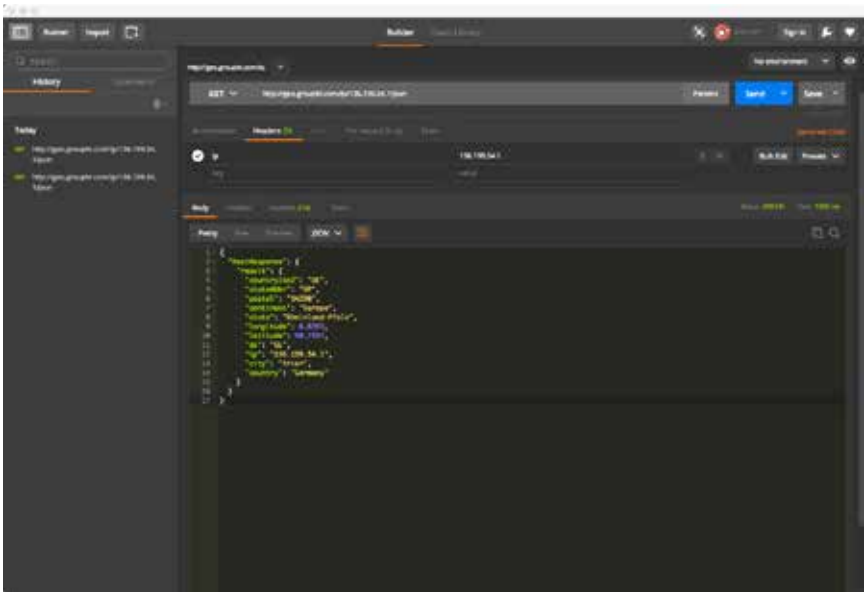
The same text expressed as XML:

```
<menu id="file" value="File">
  <popup>
    <menuitem value="New" onclick="CreateNewDoc()" />
    <menuitem value="Open" onclick="OpenDoc()" />
    <menuitem value="Close" onclick="CloseDoc()" />
  </popup>
</menu>
```

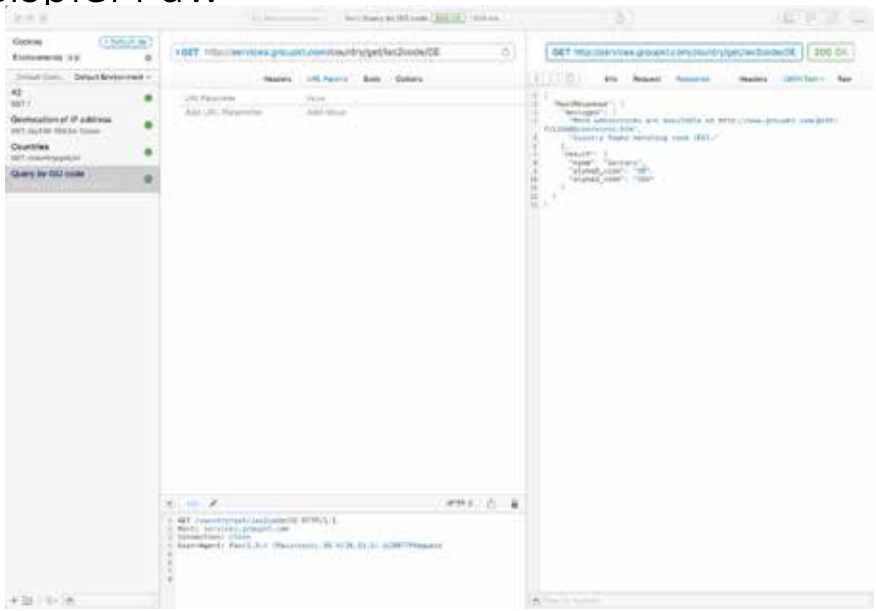
JSON Standard

16 Seiten!

Beispiel Postman

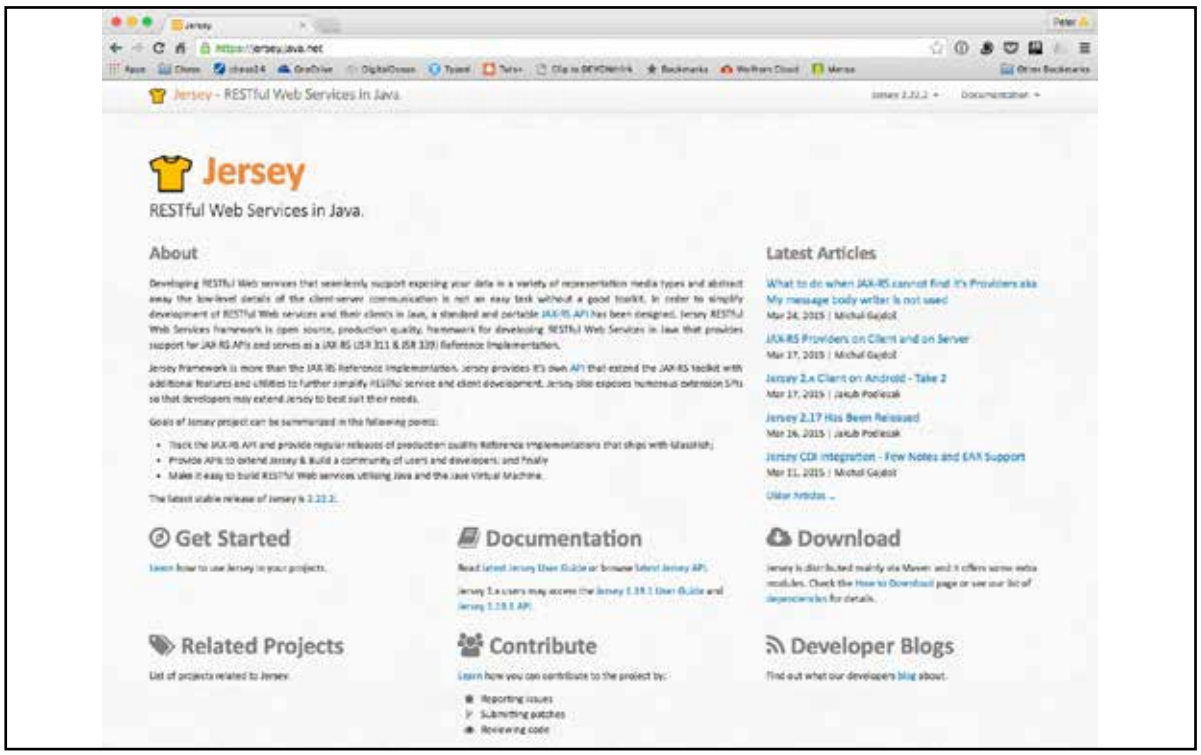


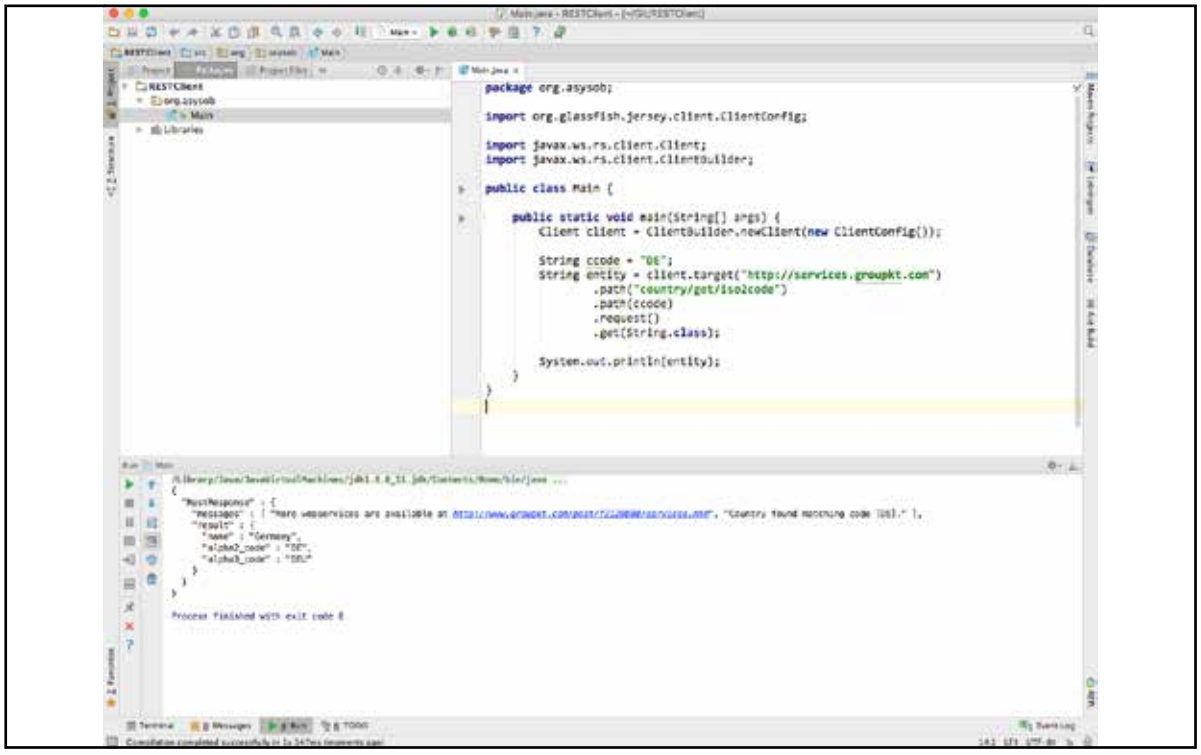
Beispiel Paw



Generische Frameworks

- Client-seitiges Framework
- Server-seitiges Framework
  - Routing
- Sprachspezifische Lösungen

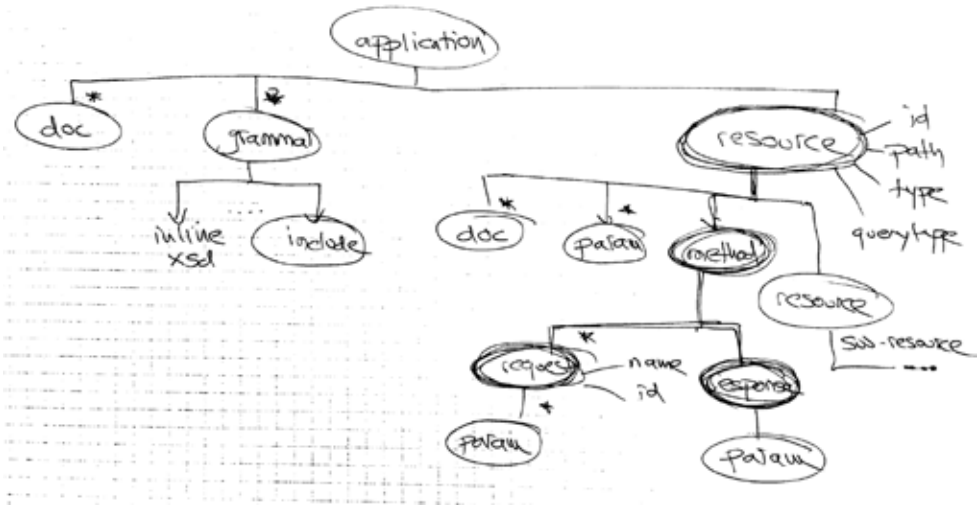

Jersey - RESTful Web Services in Java
Jersey
RESTful Web Services in Java.
About
Latest Articles
Get Started
Documentation
Download
Related Projects
Contribute
Developer Blogs


package org.asysob;
import org.glassfish.jersey.client.ClientConfig;
import javax.ws.rs.client.Client;
import javax.ws.rs.client.ClientBuilder;
public class Main {
public static void main(String[] args) {
Client client = ClientBuilder.newClient(new ClientConfig());
String ccode = "DE";
String entity = client.target("http://services.groupkt.com")
.path("country/get/iso2code")
.path(ccode)
.request()
.get(String.class);
System.out.println(entity);
}
}

WADL

- Web Application Description Language
  - XML
  - HTTP-basierte Webanwendungen
- Keine Standardisierungspläne seitens W3C
- REST-Äquivalent von WSDL
- REST ginge auch in WSDL

Struktur



Beispiel

```
<application xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://wadl.dev.java.net/2009/02 wadl.xsd"
 xmlns:tns="urn:yahoo:yn"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:yn="urn:yahoo:yn"
 xmlns:ya="urn:yahoo:api"
 xmlns="http://wadl.dev.java.net/2009/02">
  <grammars>
    <include
      href="NewsSearchResponse.xsd"/>
    <include
      href="Error.xsd"/>
  </grammars>

  <resources base="http://api.search.yahoo.com/NewsSearchService/V1/">
    <resource path="newsSearch">
      <method name="GET" id="search">
        <request>
          <param name="appid" type="xsd:string"
            style="query" required="true"/>
          <param name="query" type="xsd:string"
            style="query" required="true"/>
          <param name="type" style="query" default="all">
            <option value="all"/>
            <option value="any"/>
            <option value="phrase"/>
          </param>
          <param name="results" style="query" type="xsd:int" default="10"/>
          <param name="start" style="query" type="xsd:int" default="1"/>
          <param name="sort" style="query" default="rank">
            <option value="rank"/>
            <option value="date"/>
          </param>
          <param name="language" style="query" type="xsd:string"/>
        </request>
        <response status="200">
          <representation mediaType="application/xml"
            element="yn:ResultSet"/>
        </response>
        <response status="400">
          <representation mediaType="application/xml"
            element="ya:Error"/>
        </response>
      </method>
    </resource>
  </resources>
</application>
```

http://www.w3.org/Submission/wadl/

Reflective REST APIs

- HATEOAS
  - Hypermedia as the Engine of Application State
  - Ebenfalls Roy Fielding
- Dynamische Schnittstellenbeschreibung
- Entry points
  - Dynamic API discovery

und weiter ...