Freie Universität Berlin

Service oriented Architecture and Web Services

Netzprogrammierung
(Algorithmen und Programmierung V)

Review

What have we discussed so far?

Addressed topics so far

Descriptive models for distributed system design

- Physical model
- Architectural model
  - Architectural elements
    - Communicating entities
      - Processes
      - Objects
      - Components
      - Web Services
    - Communication paradigm
      - Inter-process communication
        - UDP sockets
        - TCP sockets
        - Multi-cast
      - Indirect communication
      - Remote invocation
    - Roles and responsibilities
      - Architectural styles
        - Client-server
        - Peer-to-peer
    - Placement
      - Multiple server
      - Proxy/Cache
      - Mobile code
  - Architectural patterns
    - Vertical distribution
      - Multi-tier
      - Thin/Fat Client
    - Horizontal distribution
- Interaction model
  - Interaction model
  - Failure model
  - Security model

Our topics today

- Web Services definition and motivating example
- Service Oriented Architecture
- Realizing web services with SOAP
  - Web Service Description Language (WSDL)
  - Universal Description Discovery & Integration (UDDI)
- RESTful web services

Service oriented Architecture and Web Services

Web Services

Defining a web service

Generic definition

- *Any application accessible to other applications over the Web.*

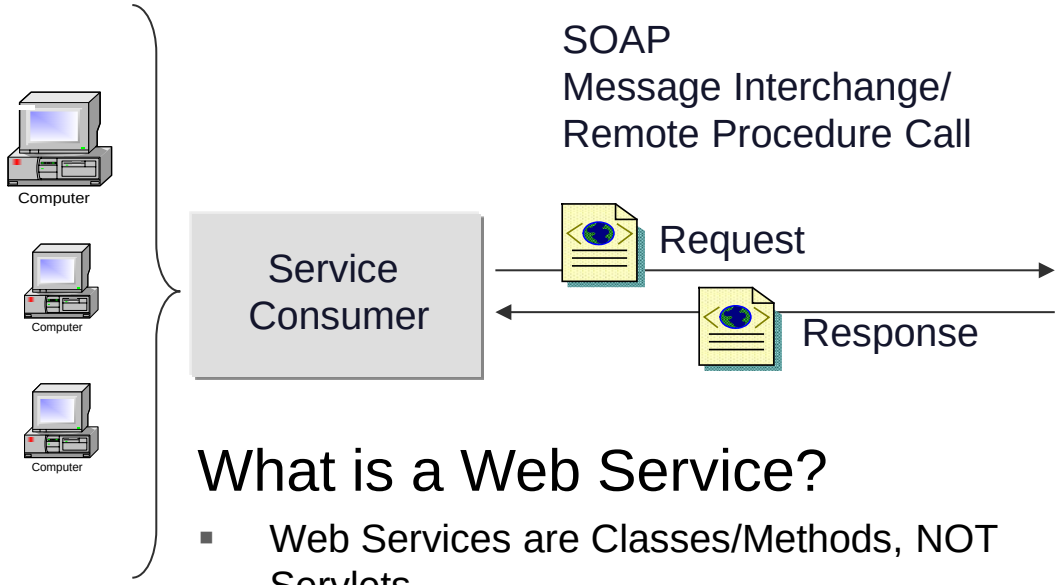
Definition of the UDDI consortium (http://www.uddi.org/pubs/UDDI_Executive_White_Paper.pdf)

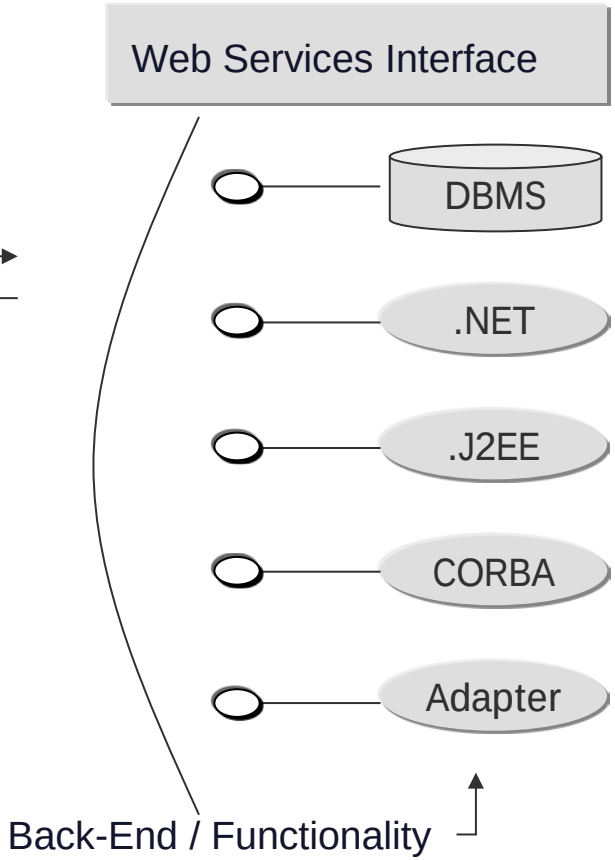
- *Web services are self-contained, modular business applications that have open, Internet-oriented, standards-based interfaces.*

Definition of the W3C (http://www.w3.org/TR/ws-arch/)

- *A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards.*

Web Services





What is a Web Service?

- Web Services are Classes/Methods, NOT Servlets
- **Loosely-coupled**
- Encapsulate functionality to logical entities
- Reuse of code and functionality
- Distributed Architecture
- Standardized interface & established Internet protocols

Characteristics of a web service

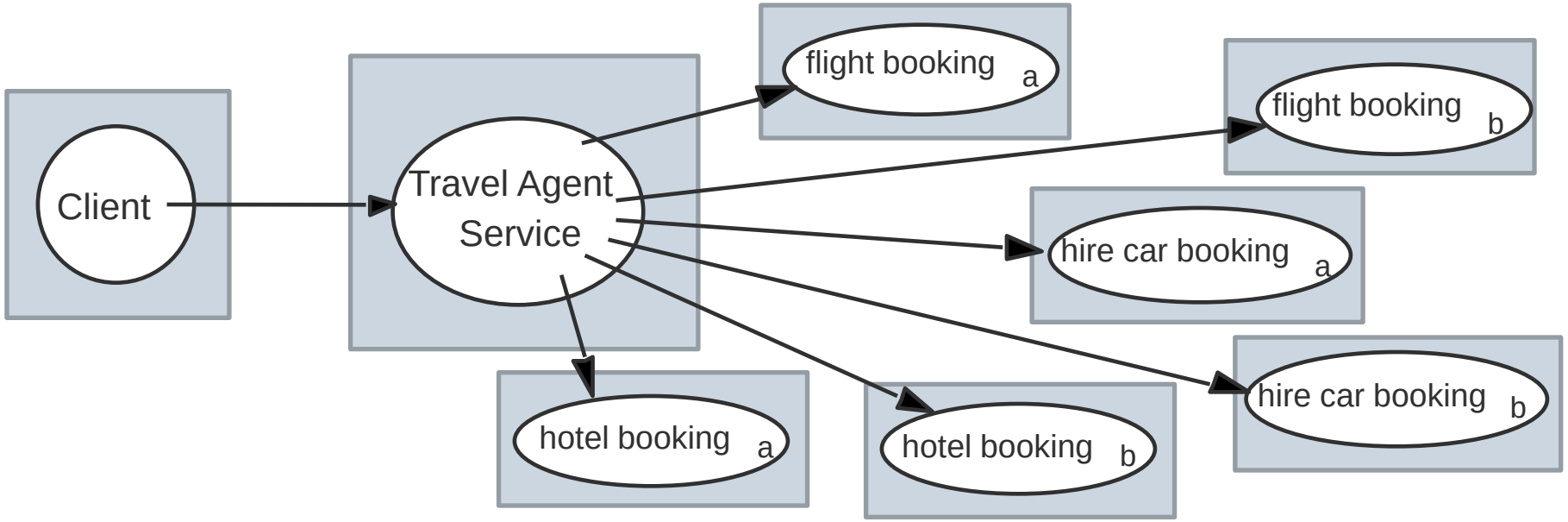
A web service interface generally consists of a collection of operations that can be used by a client over the Internet. The operations in a web service may be provided by a variety of different resources, for example, programs, objects, or databases.

The key characteristic of (most) web services is that they can process XML-formatted SOAP messages. An alternative is the REST approach.

Each web service uses its own service description to deal with the service-specific characteristics of the messages it receives.

Commercial examples include Amazon, Yahoo, Google and eBay.

The “travel agent service” example

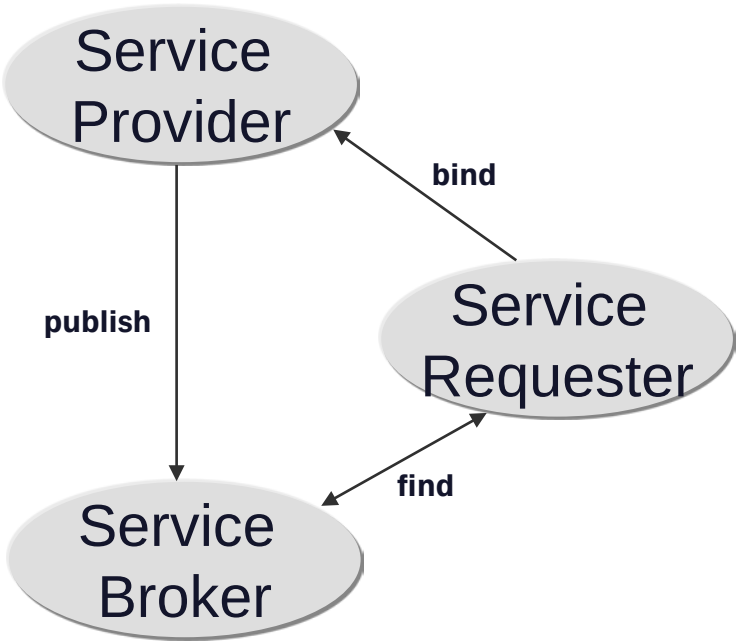


SOA – Service Oriented Architecture

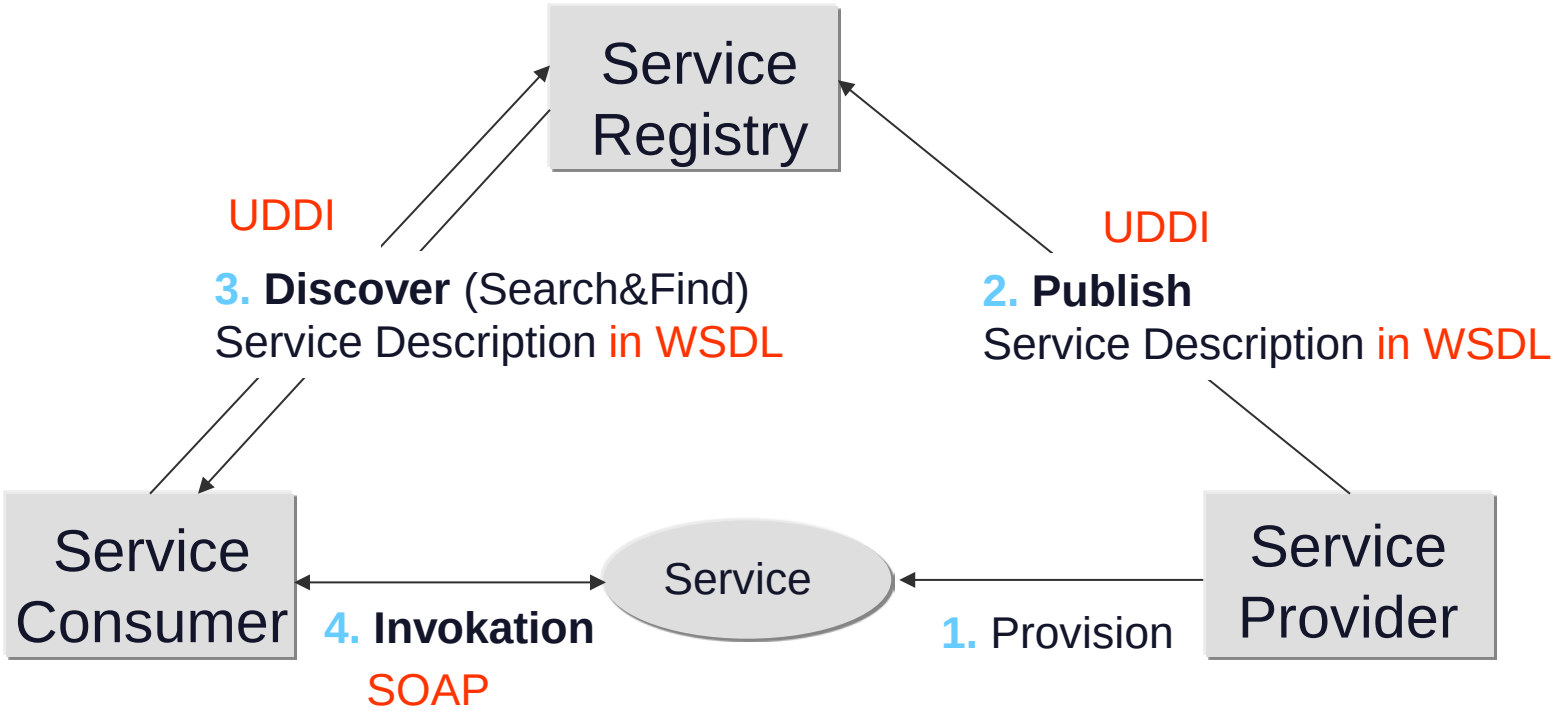
Service provider publishes service description (WSDL), e.g. on a **service broker**

Service Requester finds service (on service broker) and dynamically binds to service

Enables ad-hoc collaboration and Enterprise Application Integration (EAI) within web-based information systems



SOA: Roles and Activities using WS



Realization by WS-Architecture

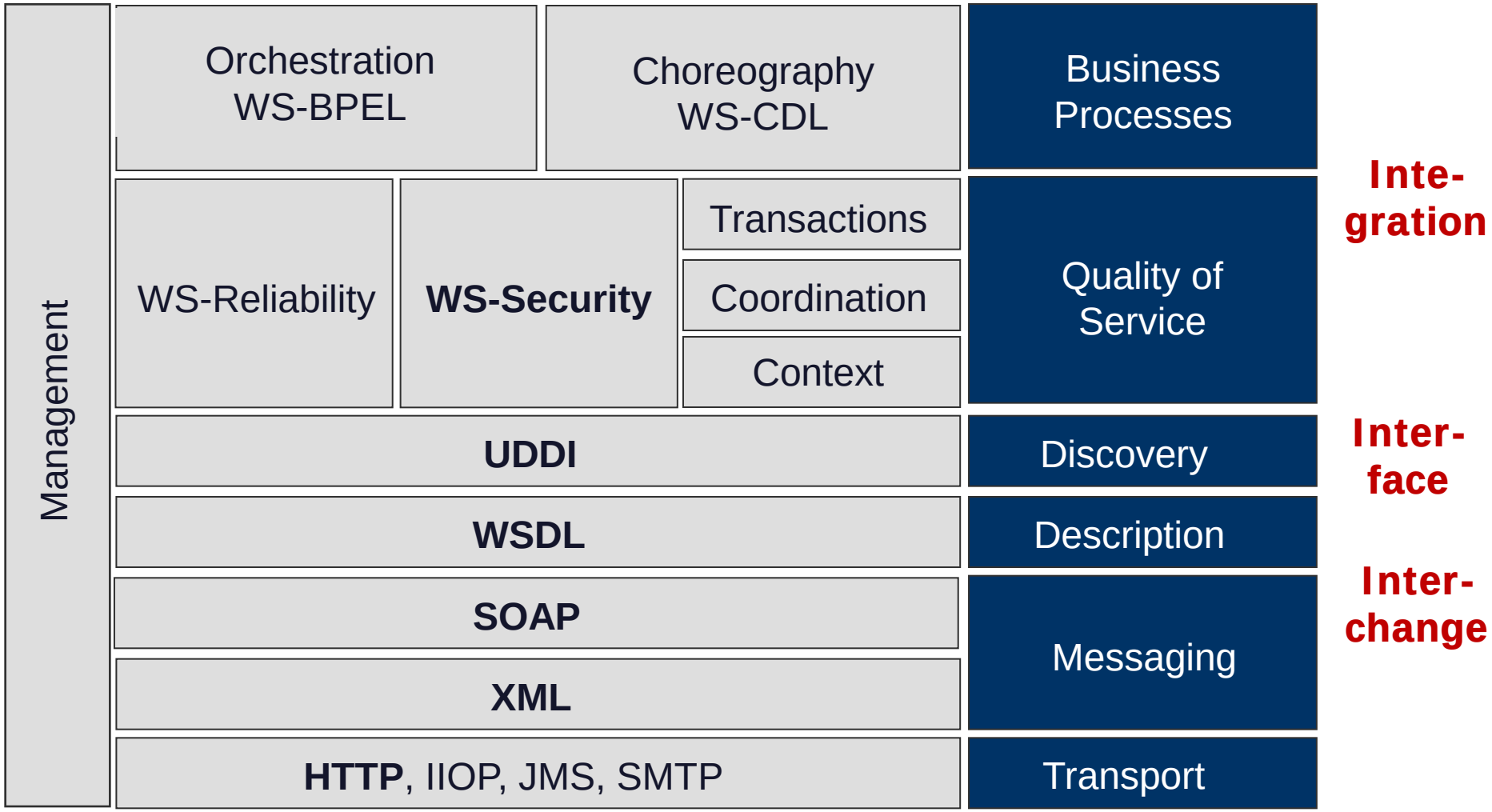
Software as a (Web) Service

- SOA - Service-oriented Architecture
  - 3 Basic Concepts: Service Description,
    Service Publication and Discovery, Service Invocation
- WS(A) - Web Services Architecture
  - SOAP (Invocation)
  - WSDL (Web Service Description)
  - UDDI (Publish, Discovery (Search and Find))

Distinction SOA-WS

- SOA not necessarily WS technology based (others e.g. Corba, RMI, …)
- WS technology not necessarily SOA based

WS-Stack und -Standards



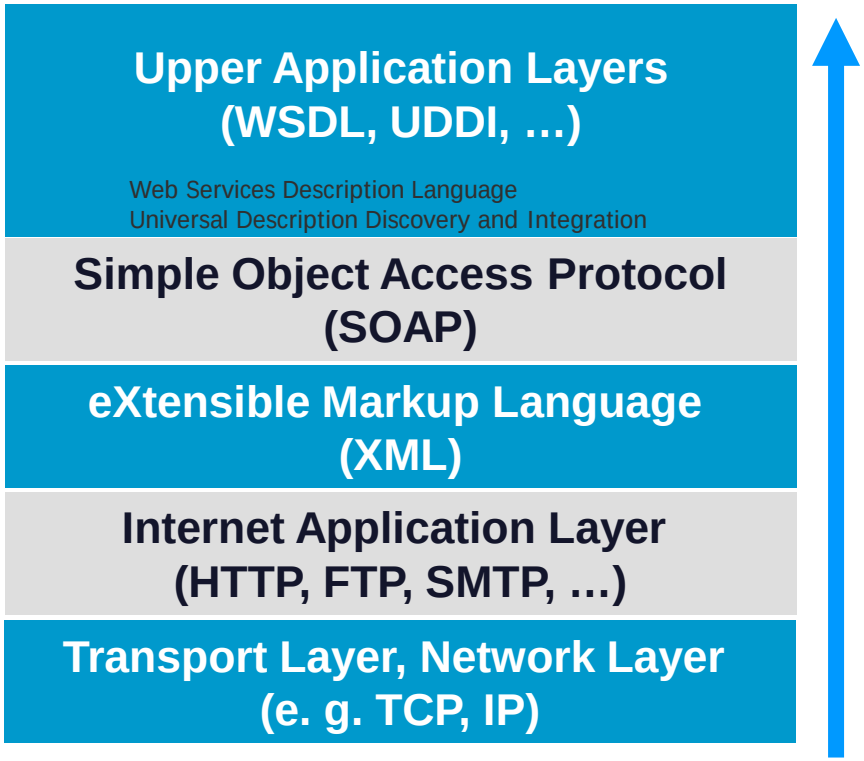
Web Services – Protocol Layers

Established Basis Layers:
- TCP/IP
- HTTP
- XML

SOAP is based on these layers

UDDI and WSDL use SOAP

Additional concepts:
- Security
- Authentification
- Quality of Service
- Session Management
- Transaction Management



Web Services

Realizing web services with SOAP

Simple Object Access Protocol (SOAP)

SOAP is designed to enable both client-server and asynchronous interaction over the Internet. It defines a scheme for using XML to represent the contents of request and reply messages as well as a scheme for the communication of documents.

It is used for information exchange and RPC, usually (but not necessarily) over HTTP.

(Very) basic SOAP architecture:

SOAP Overview I

Simple Object Access Protocol (SOAP)

Standardized by W3C

Simple **XML-based protocol** for message interchange

Not connection oriented
- Request – Response model with network interconnection

typically **RPC-like access (remote procedure call)**
- Synchronous communication – XML structure predefined

- Asynchronous communication possible
  - Not RPC but „Messaging“
  - Arbitrary XML document as SOAP payload
  - Processing of messages under control of the server

SOAP Overview II

Uses standardized Internet protocols such as HTTP, SMTP, FTP

Important goal: **Interoperability** between different heterogeneous information systems

- Currently, no (standardized) support for security concepts and transactions (but e.g. WS Security standardization ongoing)

Effort for transport and processing of data

- „complex“ XML-Format (>traffic on the wire)
- Usage of XML parser at the endpoints (> processing overhead)

No information, how SOAP should be processed (e.g. which XML parser etc.)

Transparency of SOAP transport (usage of parsers at the endpoints)

SOAP Specification

SOAP v. 1.2, W3C Recommendation (06/03):

SOAP Messaging

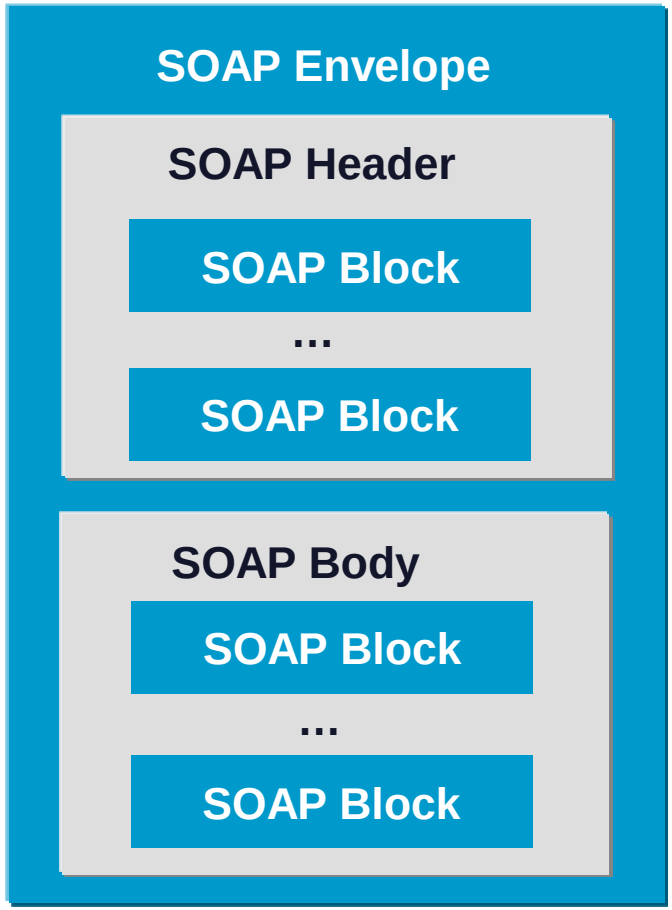
- Structure of SOAP message

SOAP Encoding

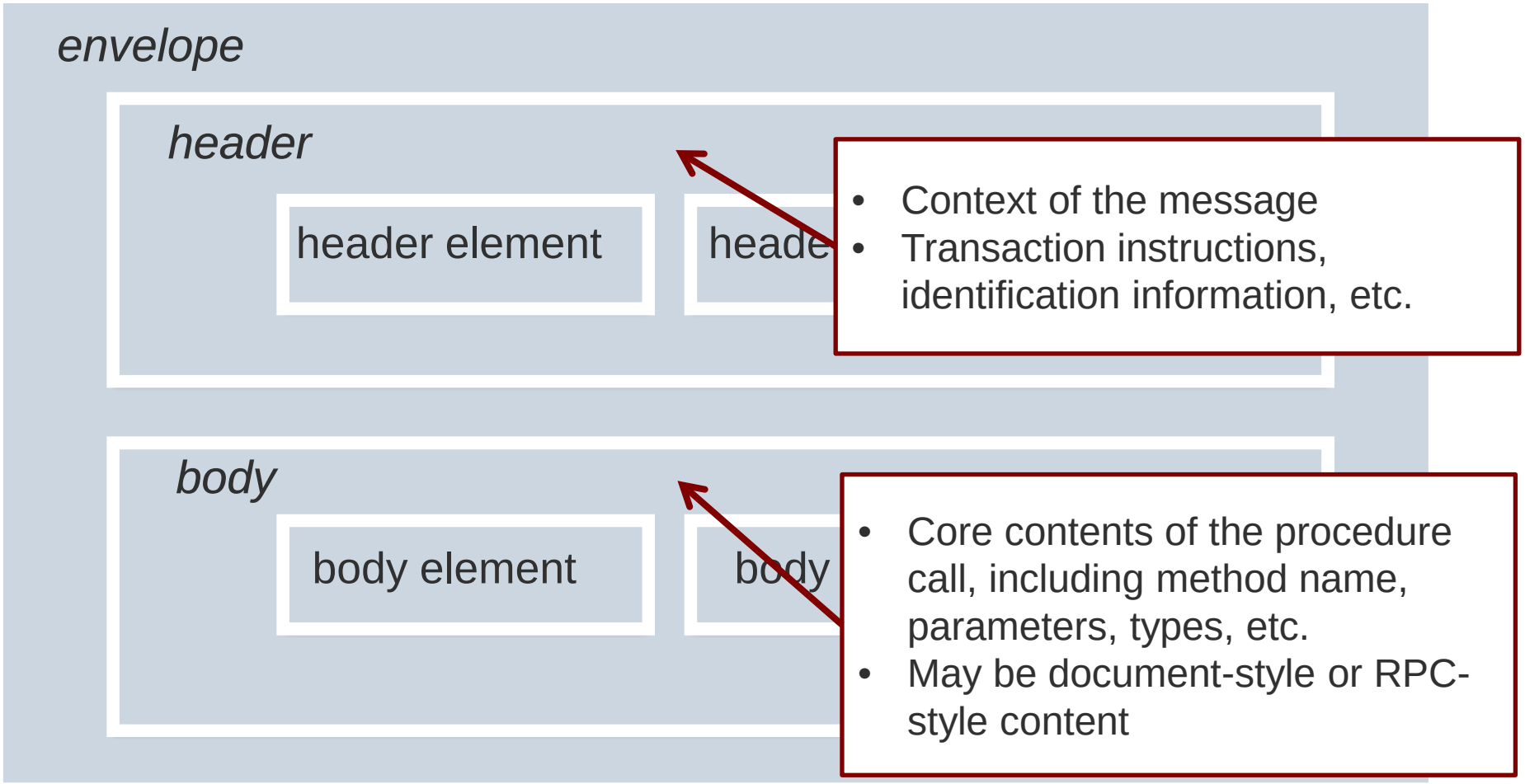
- Deserialisation rules (Encoding of data types)

SOAP Protocol Bindings

- How to bind to a particular transport protocol (e.g. HTTP)



SOAP message in an envelope



SOAP Request

```
POST /cswservice/cswWS HTTP/1.1
Host: mysoapserver
Content-Type: text/xml; charset=utf-8
…

<SOAP-ENV:Envelope
 xmlns:SOAP-ENV=„http://www.w3.org/2003/05/soap-envelope/“>

 <SOAP-ENV:Header>
    <t:TransactionCode xmlns:t=„my-URI“ xsi:type=„xsd:int“
       SOAP-ENV:mustUnderstand=„1“
       SOAP-ENV:role=„http://einserver/actor/TPMonitor “>
       156533245
    </t:TransactionCode>
 </SOAP-ENV:Header>

 <SOAP-ENV:Body>
    <method:getNextCoord xmlns:method=„my-URI2“>
       <method:xCoord>15</method:xCoord>
       <method:yCoord>124</method:yCoord>
    </method:getNextCoord>
 </SOAP-ENV:Body>

</SOAP-ENV:Envelope>
```

HTTP Header

Namespace with SOAP vocabulary, defined with SOAP version

Header with additional information; e.g. trans-actions code

Body contains the user data; here the parameters for RPC

SOAP Response

```
<SOAP-ENV:Envelope
            xmlns:SOAP-ENV=„http://www.w3.org/2003/05/soap-envelope“
            SOAP-ENV:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

  <SOAP-ENV:Body xmlns:m="http://myur">
     <m:getNextCoordResponse>
        <m:coord>123</m:coord>
     </m:getNextCoordResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Standard encoding based on XML Schema

Encoding rules for RPC

SOAP Protocol Binding

SOAP concepts transported by the underlying transport protocol (HTTP, SMTP, FTP, …)

Identification of the message receiver is done by the transport protocol

HTTP

- Eventually by Host-Header field-access identification of the target server (path resolving & dispatching by e.g. Web server, see Java-Servlets)
- Broadly adopted and accessible (firewall tolerance)
- Simple to implement
- Supports for security mechanisms (HTTPS)

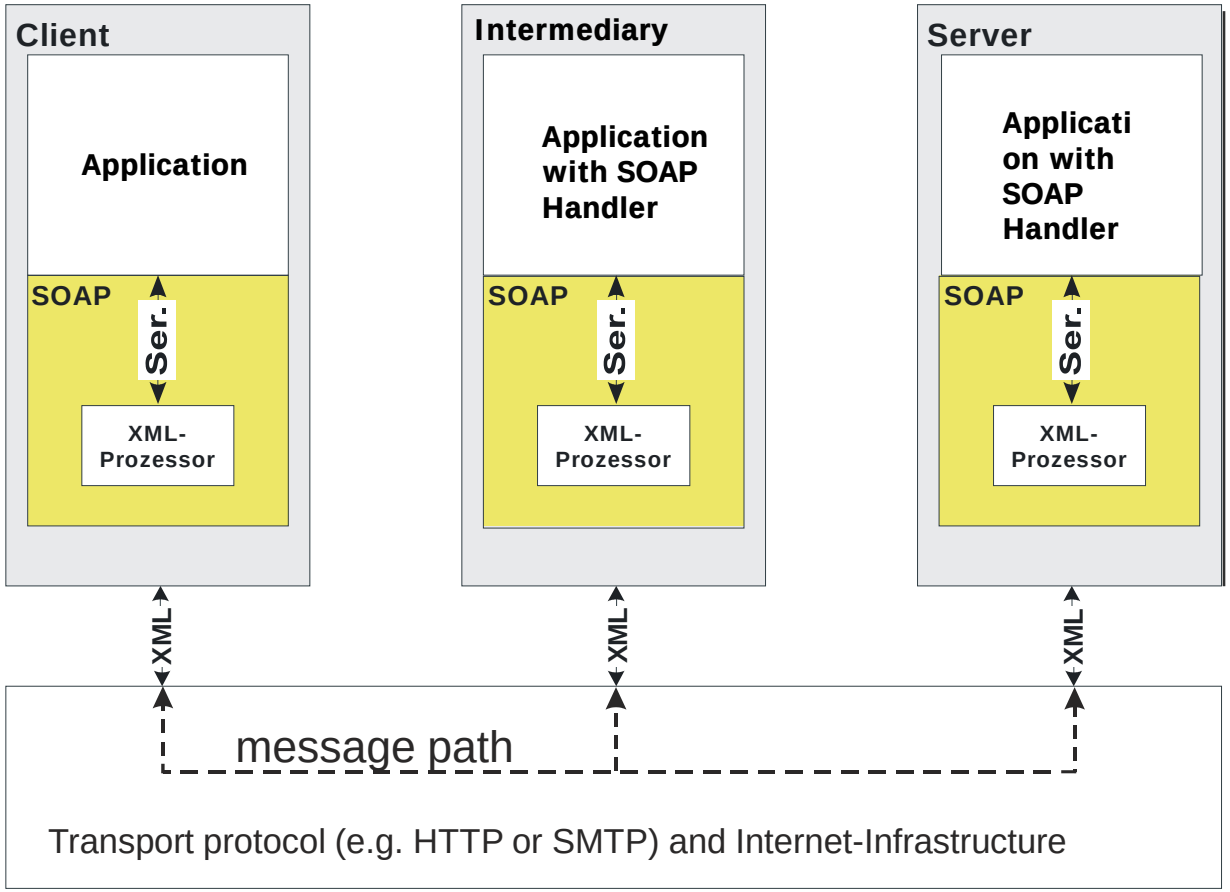
HTTP Binding

```
POST /Reservations HTTP/1.1
Host: travelcompany.example.org
SOAPAction=„Intermediary-URI“
Content-Type: application/soap+xml; charset="utf-8"
Content-Length: nnnn

<?xml version='1.0' ?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope"
>
 <env:Header>
   <t:transaction
…
```

HTTPS: Secure environment for SOAP messages

SOAP Message Transportation



Quelle: Huemer: Web Services

SOAP for RPCs

RPC as common application domain for SOAP

RPC data are transported in the SOAP body

- URI of the receiver node (in the HTTP header)
- Function-/Method name
- Function-/Method signature
- Functions parameter

RPC Request

- Structure with the name of the function
- Function parameter need to be typed and named
- Order of the parameter corresponds with the interface
- Input- as well as Input/Output parameter are possible

SOAP-RPC Example

Request

```
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
  <env:Body >
    <tmp:getTemp xmlns:tmp="http://csw.inf.fu-berlin.de/temp" >
        <tmp:plz>4040</tmp:plz>
        <tmp:strasse>Altenbergerstrasse</tmp:strasse>
        <tmp:hsnr>74</tmp:hsnr>
    </tmp:getTemp>
</env:Body></env:Envelope>
```

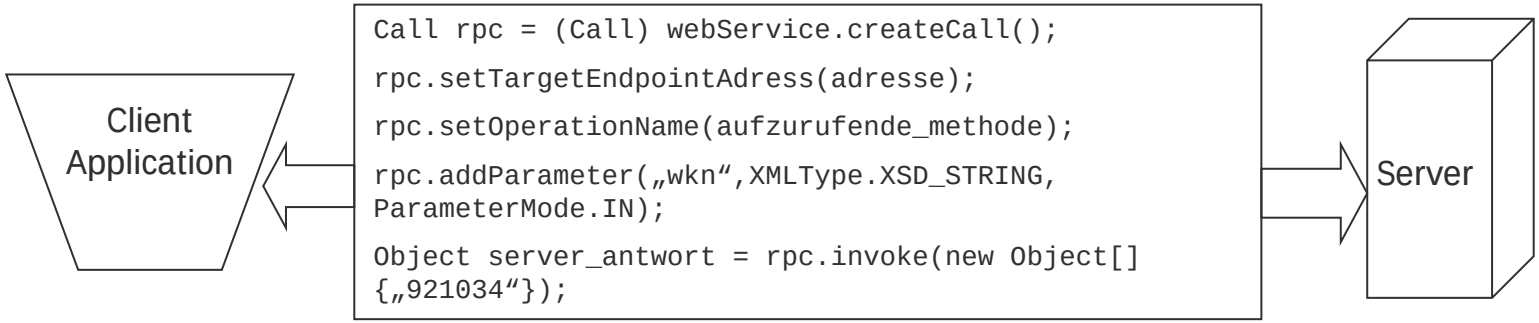
Order of parameter corresponds to the interface

Response

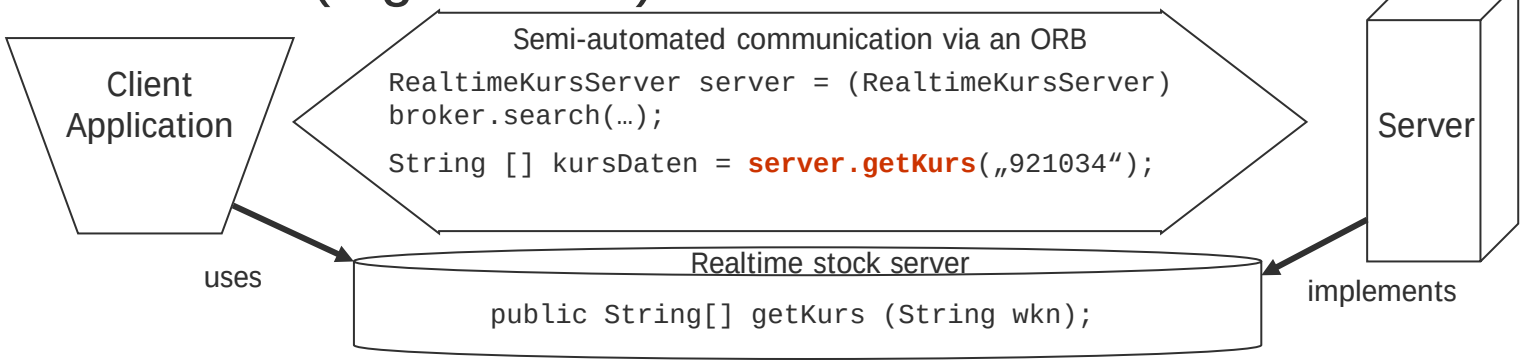
```
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
  <env:Body >
    <tmp:getTempResponse>xmlns:tmp="http://csw.inf.fu-berlin.de/temp">
      <tmp:datumZeit>10.12.2001</tmp:datumZeit>
      <tmp:tempCelsius>-3</tmp:tempCelsius>
    </tmp:getTempResponse>
  </env:Body>
</env:Envelope>
```

SOAP/RPC vs. Classic Middleware

SOAP/RPC



Middleware (e.g. CORBA)



BUT: SOAP not just as RPC protocol, but for message interchange in general

Describing a SOAP service

The **Web Service Description Language (WSDL)** provides a formal description of a web service, much like CORBA's IDL. The WSDL file is all you need to know how to call the web service; Toolkits can generate proxy code from a WSDL file directly.

Essentially, a WSDL document describes three properties of a Web Service:

- ***What* a service does** - the functions which the service can provide, and the arguments needed to invoke them.
- ***How* a service is accessed** - details of data formats and protocols required.
- ***Where* a service is located** - details of a protocol-specific network address, such as a URL.

The official WSDL definition is at http://www.w3.org/TR/wsdl.

WSDL general structure

<definitions>	Defines the namespaces used by the WSDL document to describe services.
<types>	Container for datatype definitions. XSD Schema is used for describing types.
<message>	Defines the data being exchanged. Lists data input and output
<portType>	Describes the set of operations that each port provides
<binding>	Describes the transport protocol and data format details for a each port.
<port>	A single end point defined by a binding and gives the Internet address of the service
<service>	A collection of related services.
<documentation>	Human readable documentation. Provides comments anywhere inside the WSDL document and may be used to generate application specific documentation.

Contents of a WSDL 2.0 file

Which data types are transported?

- <types/> Element

abstract

Which operations and messages are supported?

- <interface/> Element

How to transport the messages?

- <binding/> Element

concrete

Where to find the service?

- <service/> Element

WSDL 2.0 – Structure

<?xml version="1.0" encoding=„utf-8" ?>

<description ...>

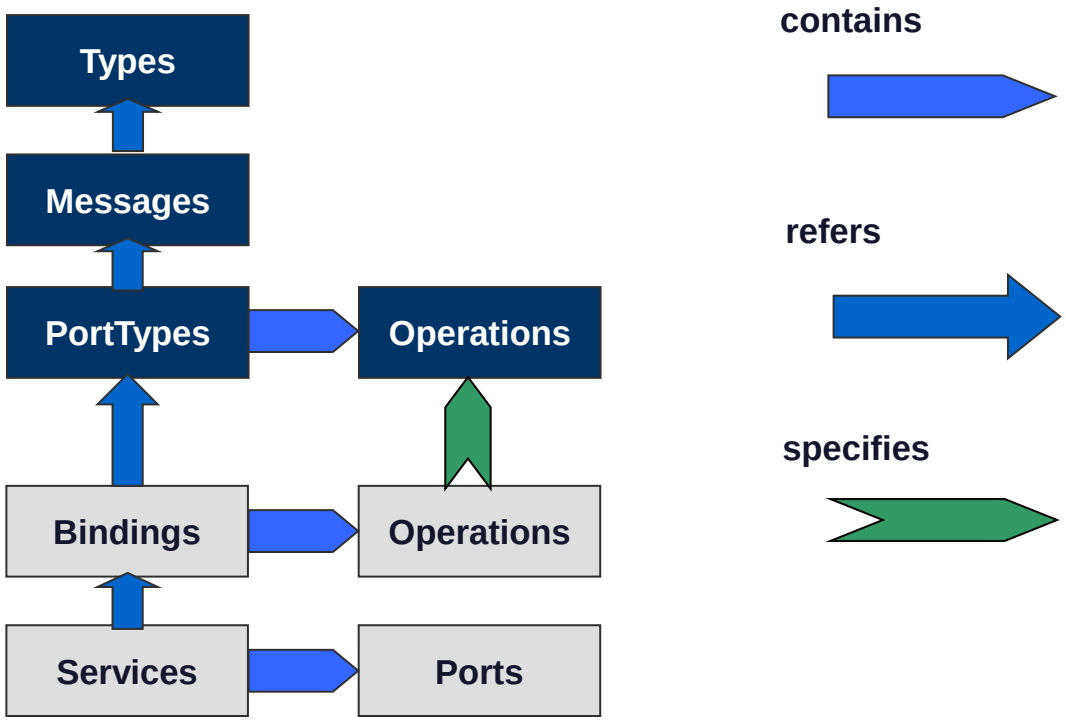
<types/>	Types
<interface/>	Interface description
<binding/>	Bindings
	Service Interface
<service/>	Service Definition
	Service Implementation

</definitions> WSDL Dokument

WSDL Document Structure

Abstract Definitions

Concrete Definitions



A directory service for use with web services

The focus of **Universal Description Discovery & Integration (UDDI)** is the definition of a set of services supporting the description and discovery of

(1) businesses, organizations, and other Web services providers,

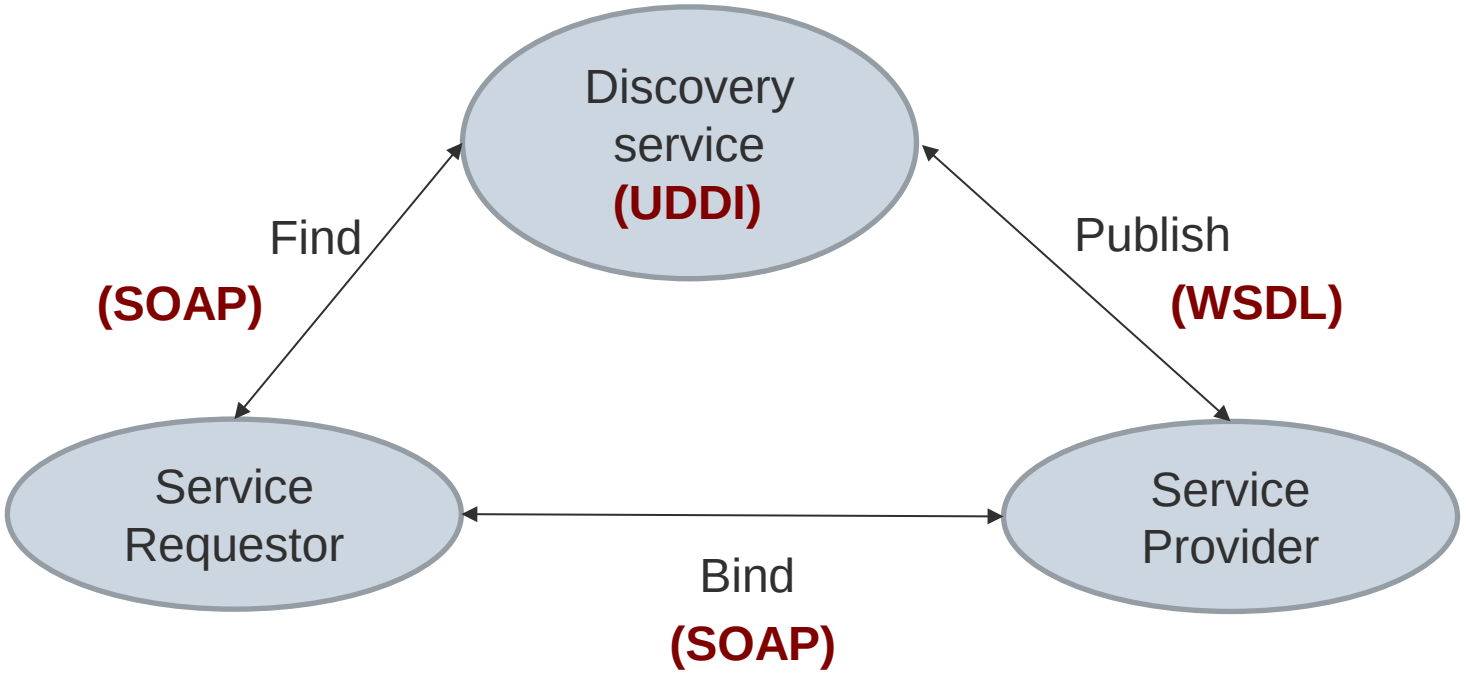
(2) the Web services they make available, and

(3) the technical interfaces which may be used to access those services.

It provides a name service and a directory service. That is, WSDL service descriptions may be looked up by name (a white page service) or by attribute (a yellow page service). They may also be accessed directly via their URLs.

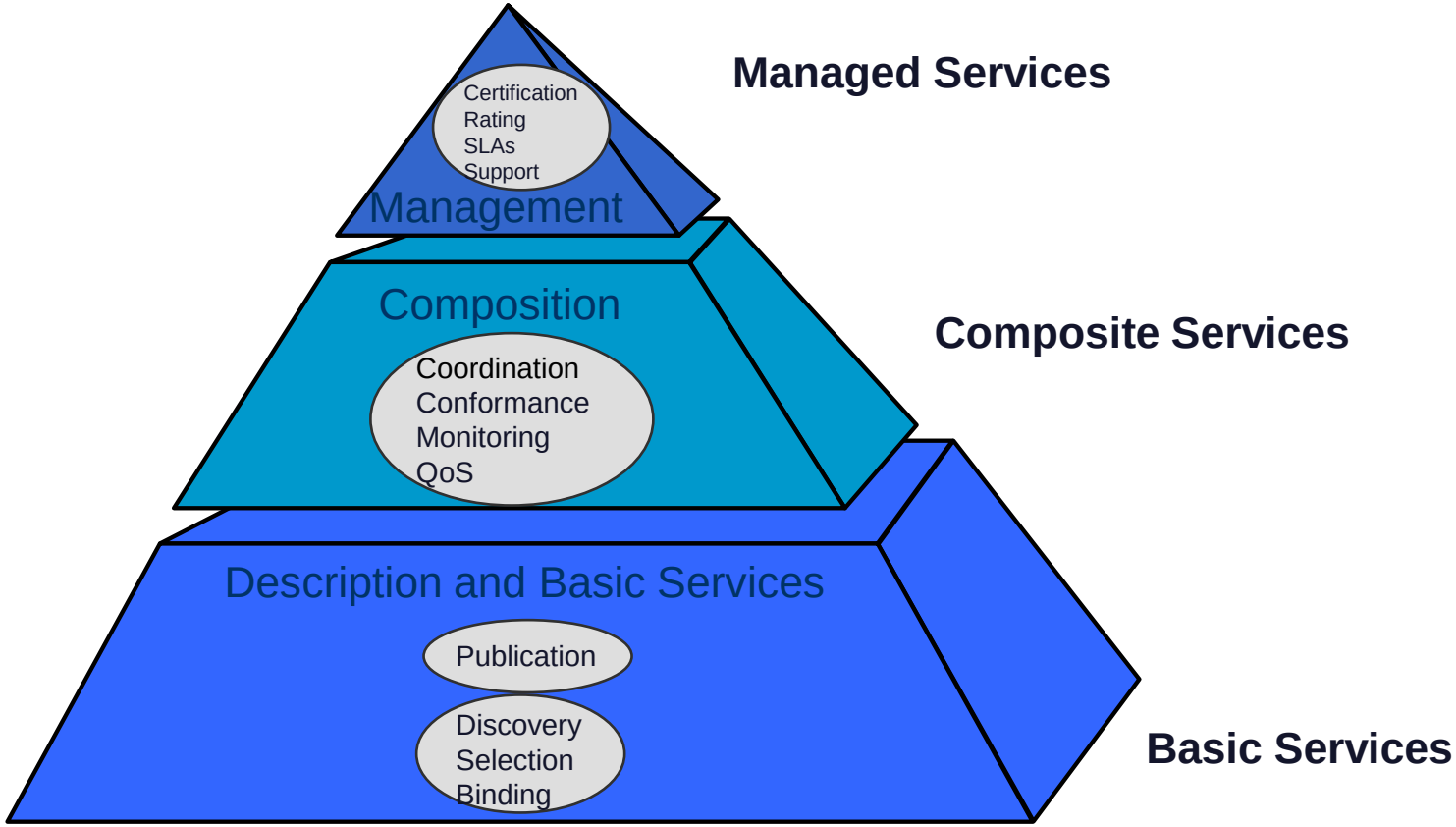
More information at: http://www.oasis-open.org/committees/uddi-spec

Web service architecture (simplified)



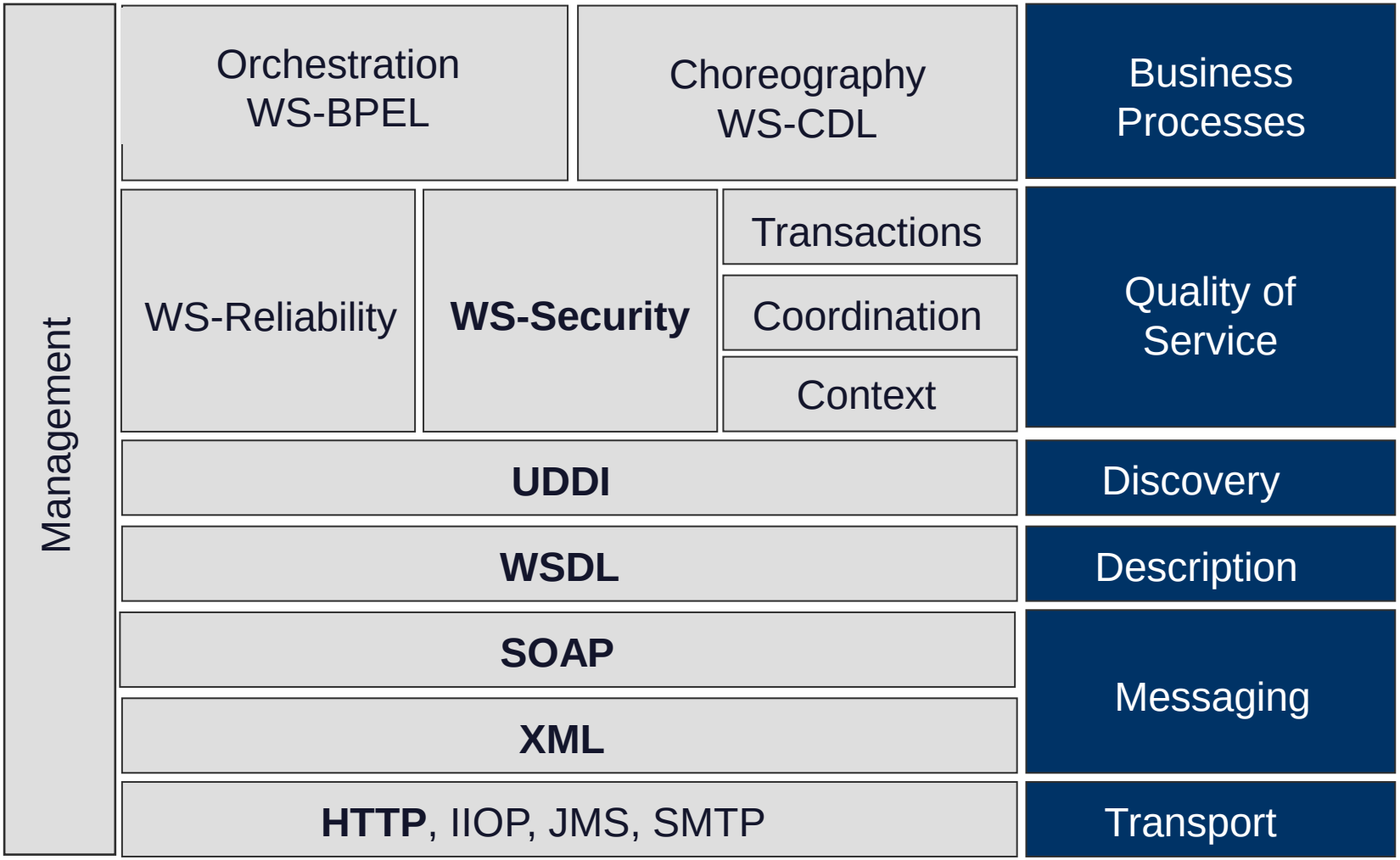
http://www.w3.org/TR/ws-arch/

Extended Service-Oriented Architecture (SOA)



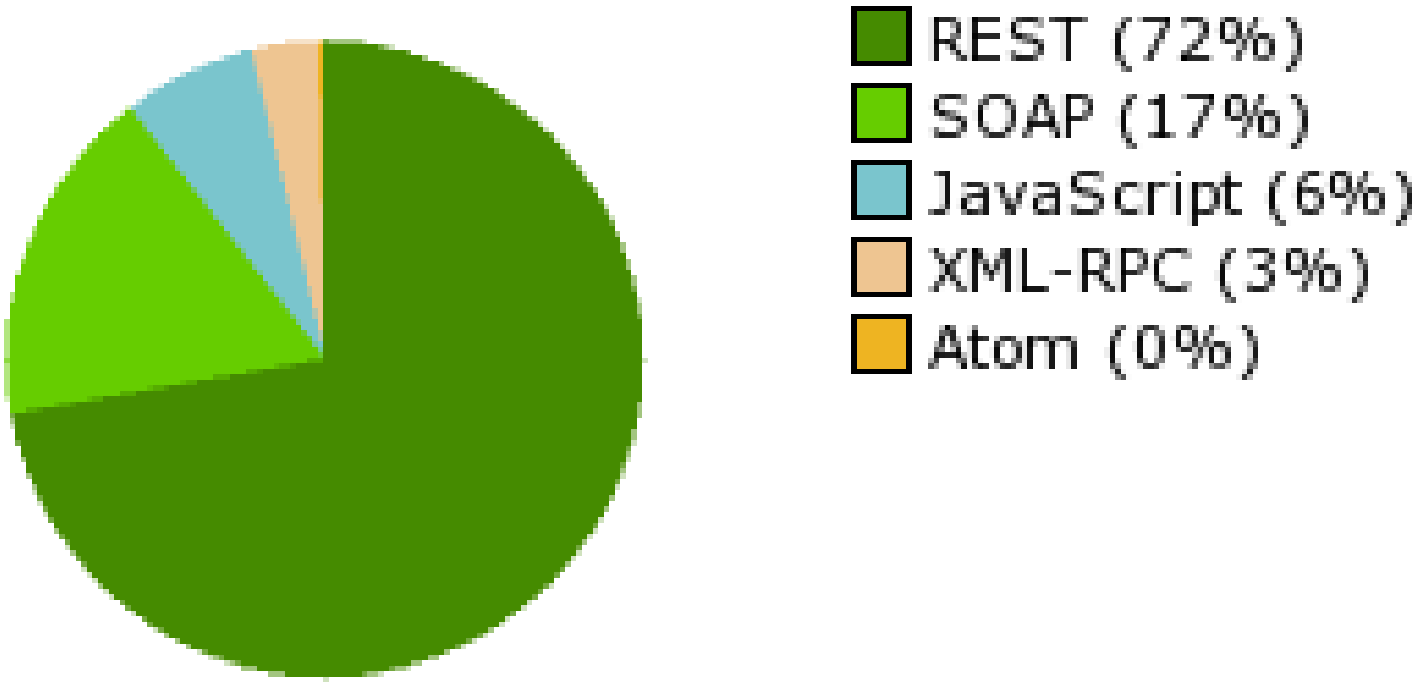
CACM, Oktober 2003/Vol. 46, No. 10

Summary: WS-Stack und -Standards




Did we talked about everything?

Protocol Usage by APIs



ProgrammableWeb.com 01/02/12

Service oriented Architecture and Web Services

Representational State Transfer (REST)

The Idea of Representational State Transfer (REST)

resource

state of the resource

the **representational** state of the Resource

What is REST?

REST stands for Representational State Transfer and it was invented by Roy Fielding in 2000.

"Representational State Transfer is intended to evoke an image of how a well-designed Web application behaves: a network of web pages (a virtual state-machine), where the user progresses through an application by selecting links (state transitions), resulting in the next page (representing the next state of the application) being transferred to the user and rendered for their use."

REST is ***an architecture style*** for designing networked applications. The idea is that, rather than using complex mechanisms such as CORBA, RPC or SOAP to connect between machines, simple HTTP is used to make calls between machines.

The World Wide Web itself, based on HTTP, can be viewed as a REST-based architecture.

(Elkstein, 2008)

REST Architecture of the Web - Data

Representational State Transfer (REST)

- **Style of software architecture** for distributed hypermedia systems such as the World Wide Web.

Resources: Abstraction from an information (e.g. „weather in Berlin“)

Resources is **a representation at a certain time** of a set of

- **representations** (e.g. HTML site with the description of the weather picture)
- **identifications** (e.g. URI / URL of the description page, the image)

Concepts of resources

- Abstraction from a particular representation;
- Allows late biding of a value to the identification

REST design principles

Stateless Client/Server Protocol: Each message contains all the information needed by a receiver to understand and/or process it. This constraint attempts to “keep things simple” and avoid needless complexity.

A set of uniquely addressable resources enabled by a universal syntax for resource identification (URI); “Everything is a Resource” in a RESTful system.

A set of well-defined operations that can be applied to all resources; In the context of HTTP, the primary methods are POST, GET, PUT, and DELETE, similar (but not exactly) to the database world's notion of CRUD (Create, Read, Update, Delete).

Resources are typically stored in a structured data format that supports hypermedia links, such as HTML or XML.

Resources

Resources are defined by URIs

- Resources can never be accessed or manipulated directly
- REST works with resource representations

Resources are all the things we want to work with

- If you cannot name something, you cannot do anything with it
- A popular resource type on the Web are documents
- Documents usually are a structured collection of information

Documents are abstract concepts of descriptive resources

- They may be used in different contexts (e.g., formats)
- Different applications may be interested in different representations
- The underlying resource is always the same

State

State is represented as part of the content being transferred

- Server interruptions do not create problems for the client
- It is possible to switch between servers for different interactions
- Clients can simply store the representation to save the state

State transfer makes the system scalable

- Data transfer is not state-specific (no stateful connection handling)
- State is transferred between client and server

Establishing a common model

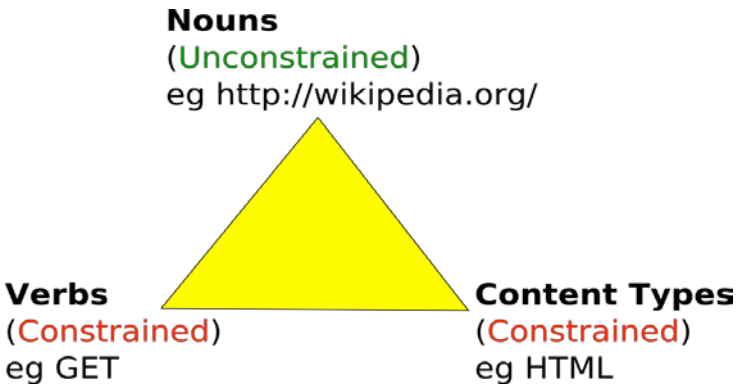
Distributed systems must be based on a shared model

- Traditional systems must agree on a common API
- REST systems structure agreement into three areas

REST is built around the idea of simplifying agreement

- *nouns* are required to name the resources that can be talked about
- *verbs* are the operations that can be applied to named resources
- *content types* define which information representations are available

REST triangle



Nouns

Nouns are the names of resources

- In most designs, these names will be URIs
- URI design is a very important part of a REST-based system design

Everything of interest should be named

- By supporting well-designed names, applications can talk about named things
- New operations and representations can be introduced

Separating nouns from verbs and representations improves extensibility

- Applications might still work with resources without being able to process them
- Introducing new operations on the Web does not break the Web
- Introducing new content types on the Web does not break the Web

Verbs

Operations which can be applied to resources.

The core idea of REST is to use universal verbs only

- Universal verbs can be applied to all nouns

For most applications, HTTP's basic methods are sufficient

GET: Fetching a resource (there must be no side-effects)

PUT: Transfers a resource to a server (overwriting if there already is one)

POST: Adds to an existing resource on the server

DELETE: Discards a resource (its name cannot be used anymore)

Corresponding to the most popular basic database operations - CRUD: Create, Read, Update, Delete

POSTing

POST adds instead of an overwriting update.

POST can have different effects

- By POSTing, state is changed and a new resource is created
- By POSTing, only the existing resource is changed
- The server signals the difference using HTTP responses (200 OK or 201 Created)

This is a design choice

- If the added information needs to be accessible individually, create a new resource
- For changes of an existing resource, no new resource has to be created

Make sure that resources are navigable using URIs

- If appropriate, a relationship can be represented in the resource format

Content types

Representations should be machine-processable

- They don't have to, they may be opaque to applications
- In many cases, machine-processable representations are advantageous

Resources are abstractions, REST passes representations around

- Resources can have various representations (i.e., content types)
- Clients can request content types they are interested in

Adding or changing content types does not change the system architecture

- Different clients and servers support different content types
- Content Negotiation allows content types to be negotiated dynamically

State management

Essential for supporting stateless interactions

Cookies are a frequently used mechanism for managing state

- In many cases used for maintaining session state (login/logout)
- More convenient than having to embed the state in every representation

Cookies have two interesting client-side side-effects

- They are stored persistently independent from any representation
- They are “shared state” within the context of one browser

Principles of REST Web Service Design

1. Identify all of the conceptual entities you want to expose as services.

2. Create a URL to each resource.

3. Categorize your resources according to whether clients can just receive a representation of the resource, or whether clients can modify (add to) the resource. For the former, make those resources accessible using an HTTP GET. For the later, make those resources accessible using HTTP POST, PUT, and/or DELETE.

4. Design to reveal data gradually. Don't reveal everything in a single response document. Provide hyperlinks to obtain more details.

5. Describe how your services are to be invoked using either a WSDL document, or simply an HTML document.

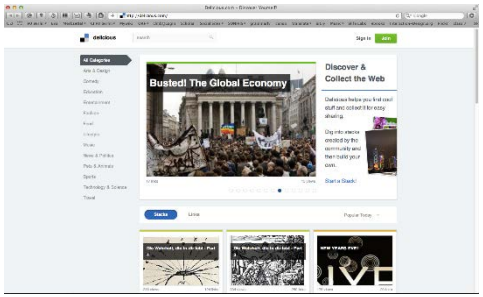
http://www.xfront.com/REST-Web-Services.html

Representational State Transfer (REST)

A RESTful Web service, an example

Please read the complete example on
http://www.peej.co.uk/articles/restfully-delicious.html

delicious



Delicious is a social bookmarking web service for storing, sharing, and discovering web bookmarks.

Delicious has “a simple REST API”, that means a simple POX over HTTP API or REST-RPC hybrid service

Delicious’s API isn't very RESTful. Why not?

- First class objects aren't exposed as resources, so bookmarks or tags can not be accessed directly.
- HTTP methods not used correctly, everything is done via GET even operations that change things.
- Resource representations not interconnected, you can't traverse from a list of bookmarks to a single bookmark.

What do we want to do with such a web service?

- Get a list of all our bookmarks and to filter that list by tag or date or limit by number
- Get the number of bookmarks created on different dates
- Get the last time we updated our bookmarks
- Get a list of all our tags
- Add a bookmark
- Edit a bookmark
- Delete a bookmark
- Rename a tag

Our two resources are bookmarks and tags:

- http://del.icio.us/api/[username]/bookmarks
- http://del.icio.us/api/[username]/tags

Get all bookmarks

URL	http://del.icio.us/api/[username]/bookmarks/	
Method	GET	
Querystring	tag=	Filter by tag
	dt=	Filter by date
	start=	The number of the first bookmark to return
	end=	The number of the last bookmark to return
Returns	200 OK & XML (delicious/bookmarks+xml)	
	401 Unauthorized	
	404 Not Found	

An example delicious/bookmarks+xml document

GET http://del.icio.us/api/peej/bookmarks/?start=1&end=2

```
<?xml version="1.0"?>
<bookmarks start="1" end="2"
   next="http://del.icio.us/api/peej/bookmarks?start=3&amp;end=4">
   <bookmark url="http://www.example.org/one" tags="example,test"
      href="http://del.icio.us/api/peej/bookmarks/a211528fb5108cddaa4b0d3aeccdbdcf"
      time="2005-10-21T19:07:30Z">
      Example of a Delicious bookmark
   </bookmark>
   <bookmark url="http://www.example.org/two" tags="example,test"
      href="http://del.icio.us/api/peej/bookmarks/e47d06a59309774edab56813438bd3ce"
      time="2005-10-21T19:34:16Z">
      Another example of a Delicious bookmark
   </bookmark>
</bookmarks>
```

HTTP & Java

You want to compose a HTTP request message in Java and then send it to a HTTP Web server?

- Java standard library http://docs.oracle.com/javase/7/docs/api/java/net/HttpURLConnection.html
- Apache HttpComponents http://hc.apache.org/
- RESTful web framework for Java http://www.restlet.org/
- JAX-RS: Java API for RESTful Web services http://www.jcp.org/en/jsr/detail?id=311

Distributed objects and components

Summary

REST vs. SOAP-based web services

REST is a description of the Web's design principles

- It is not something new, it is simply a systematic view of the Web
- REST's claim is to be able to learn from the Web's success

Web Services (in their narrow sense) do not build on REST

- They use HTTP as a transport protocol
- They re-create Web functionality through additional specifications (WS-*)
- They have been built by programmers using a top-down approach

REST and Web Services have different design approaches

- REST starts at the resources and takes everything from there
- Web Services focus on messages, which in most cases are operations

So, what have we learned today?

- SOA - Service-oriented Architecture
- WS(A) - Web Services (Architecture)

- Web Services support loosely-coupled interaction via a web service interface.
- There are different approaches to realize web services and web service architectures.
- Even though, people talk about SOAP, REST is much more applied in practice.
- Difference between SOAP- and REST-based web services.

Questions

- What is a SOA? Explain the three roles.
- What is a Web Service Architecture (WSA)? How can a WSA used to implement a SOA?
- What is SOAP? How can SOAP be used for RPCs / RMIs?
- What is WSDL? Explain the structure of WSDL.
- What is REST? How can REST be used for Web Services?

References

George Coulouris, Jean Dollimore, Tim Kindberg: *Distributed Systems: Concepts and Design*. 5th edition, Addison Wesley, 2011.

American National Standards Institute, "Coded Character Set -- 7-bit American Standard Code for Information Interchange", ANSI X3.4, 1986.

Elkstein, M.: Learn REST: A Tutorial. Blog. URL: http://rest.elkstein.org/

Wilde, Erik: Web Architecture. Lecture INFO 290-03 (CCN 42584). UC Berkeley. 2008. URL: http://dret.net/lectures/web-fall08/

Roy Thomas Fielding: Architectural Styles and the Design of Network-based Software Architectures. Dissertation. University of California Irvine. 2000.
URL: http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm

Further reading:

“How I Explained REST to My Wife” http://www.eioba.com/a/1htn/how-i-explained-rest-to-my-wife

W3C (1998): Cool URIs don't change. http://www.w3.org/Provider/Style/URI