Prof. Dr. Claudia Müller-Birn
Institute for Computer Science, Networked Information Systems

Freie Universität Berlin

Service oriented Architecture and Web Services

03-01-2012

Netzprogrammierung

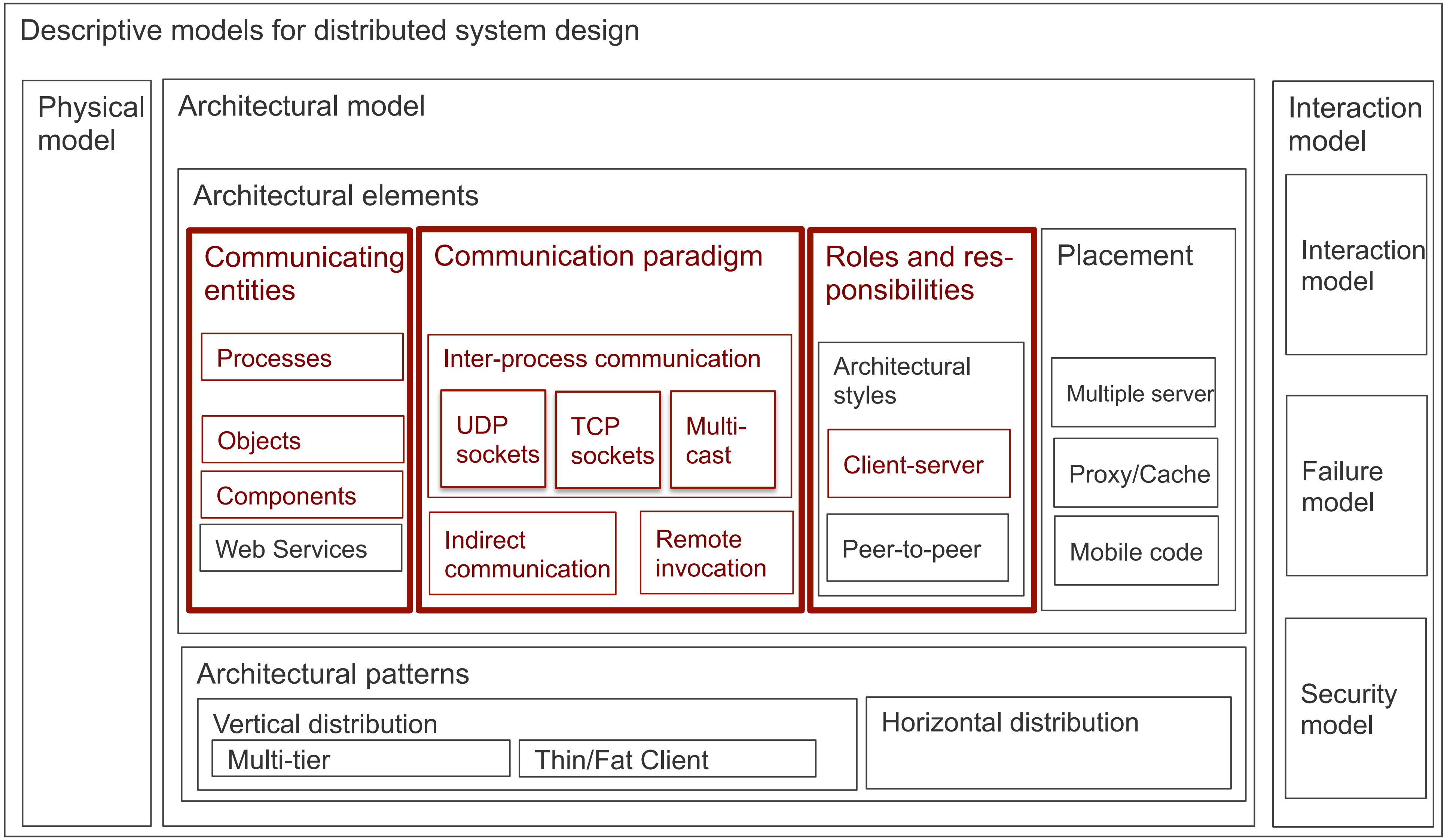
(Algorithmen und Programmierung V)

Freie Universität Berlin

Review

What have we discussed so far?

Addressed topics so far



Our topics today

- Web Services definition and motivating example

- Web service infrastructure and components
    - Uniform Resource Identifier (URI)
    - The Hypertext Transfer Protocol (HTTP)

- 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.*

Characteristics of a web service

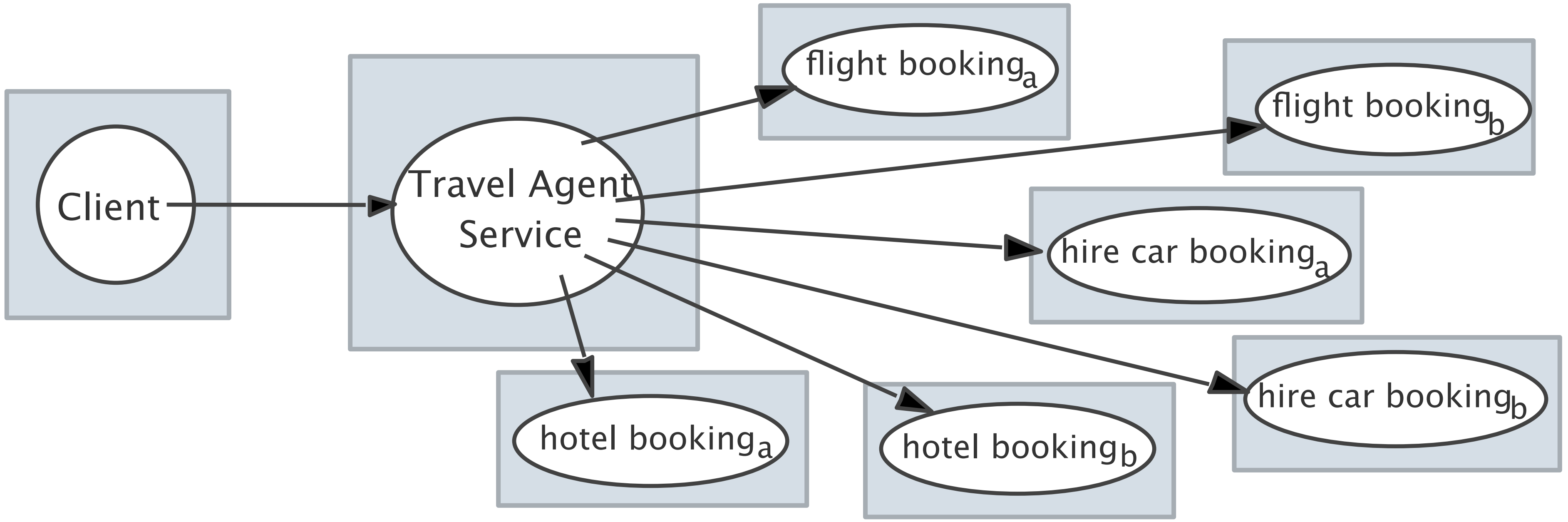
A web service interface generally consists of a collection of operations that can be used by a client over the Internet. The operations in a web service may be provided by a variety of different resources, for example, programs, objects, or databases.

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

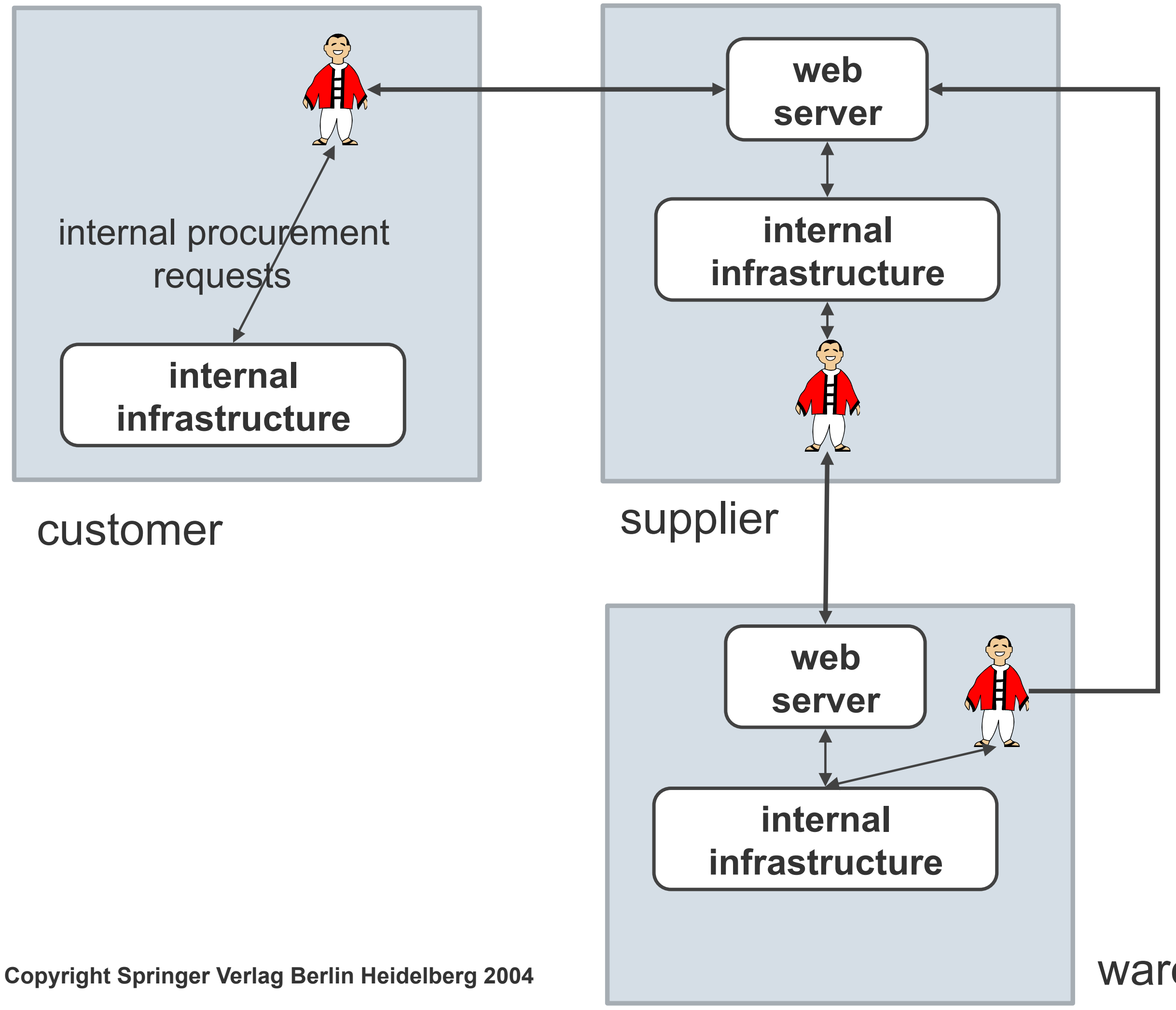
Each web service uses its own service description to deal with the service-specific characteristics of the messages it receives.

Commercial examples include Amazon, Yahoo, Google and eBay.

The “travel agent service” example

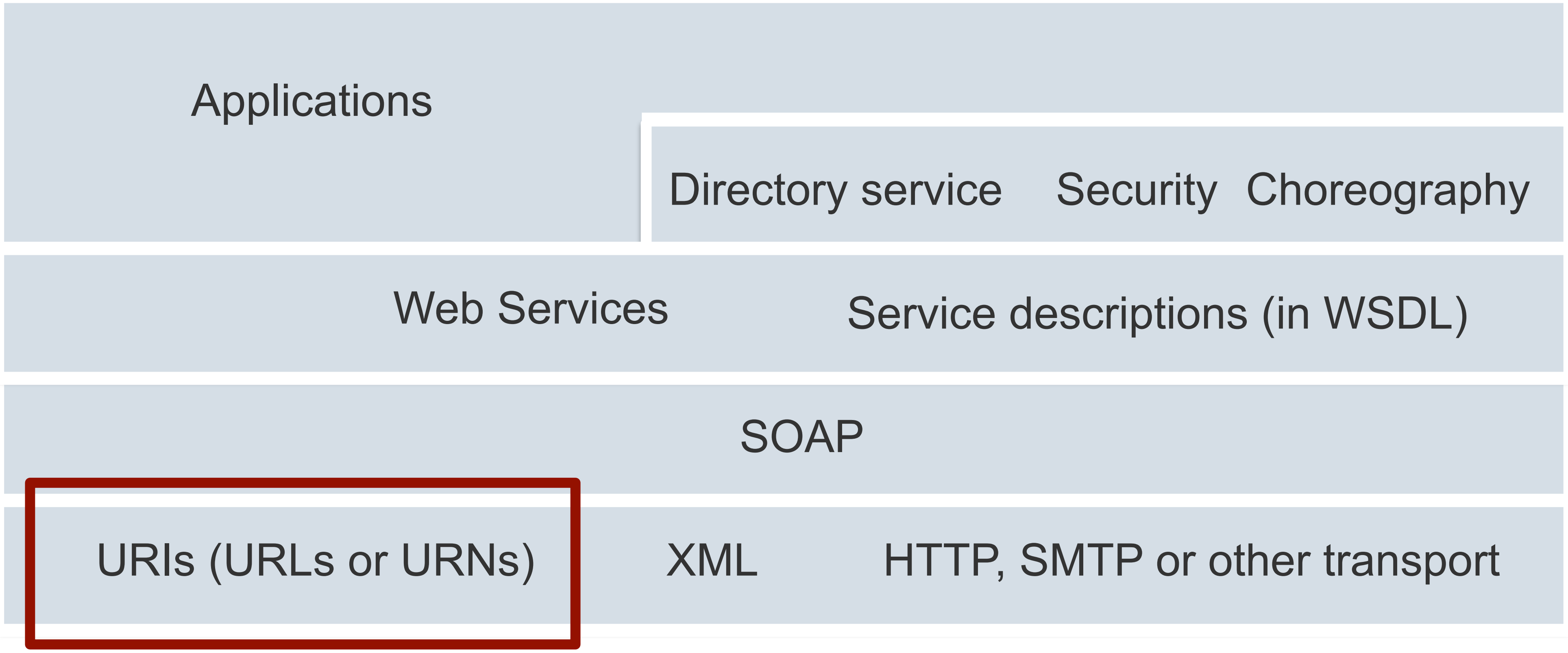


Motivating example



B2B interactions occur by accessing Web pages, filling Web forms, or via email.

Web service infrastructure and components



Web Services

Uniform Resource Identifier (URI)

Resource identification

A Uniform Resource Identifier (URI) provides a simple and extensible means for identifying a resource.

What is a resource?

- The term "resource" is used in a general sense for whatever might be identified by a URI
- Familiar examples are websites, books, places, people, relations between these resources but also abstract concepts, such as the operators and operands of a mathematical equation

The concept of an URI is already established in various domains such as the Web (URL), books (ISBN), digital object identifier (DOI).

http://www.ietf.org/rfc/rfc3986.txt

URI, URL and URN

A URI can be further classified as a locator, a name, or both.

Uniform Resource Locator (URL, RFC1738)

- Refers to the subset of URIs that, in addition to identifying a resource, provide a means of locating the resource by describing its primary access mechanism (e.g., its network "location”)
- Problem: can change over a lifetime of a web resource

Uniform Resource Name (URN, RFC2141)

- Refers to URIs, which are required to remain globally unique and persistent even when the resource ceases to exist or becomes unavailable, and to any other URI with the properties of a name

http://www.ietf.org/rfc/rfc3986.txt

Resources vs. representations

URIs identify *resources*

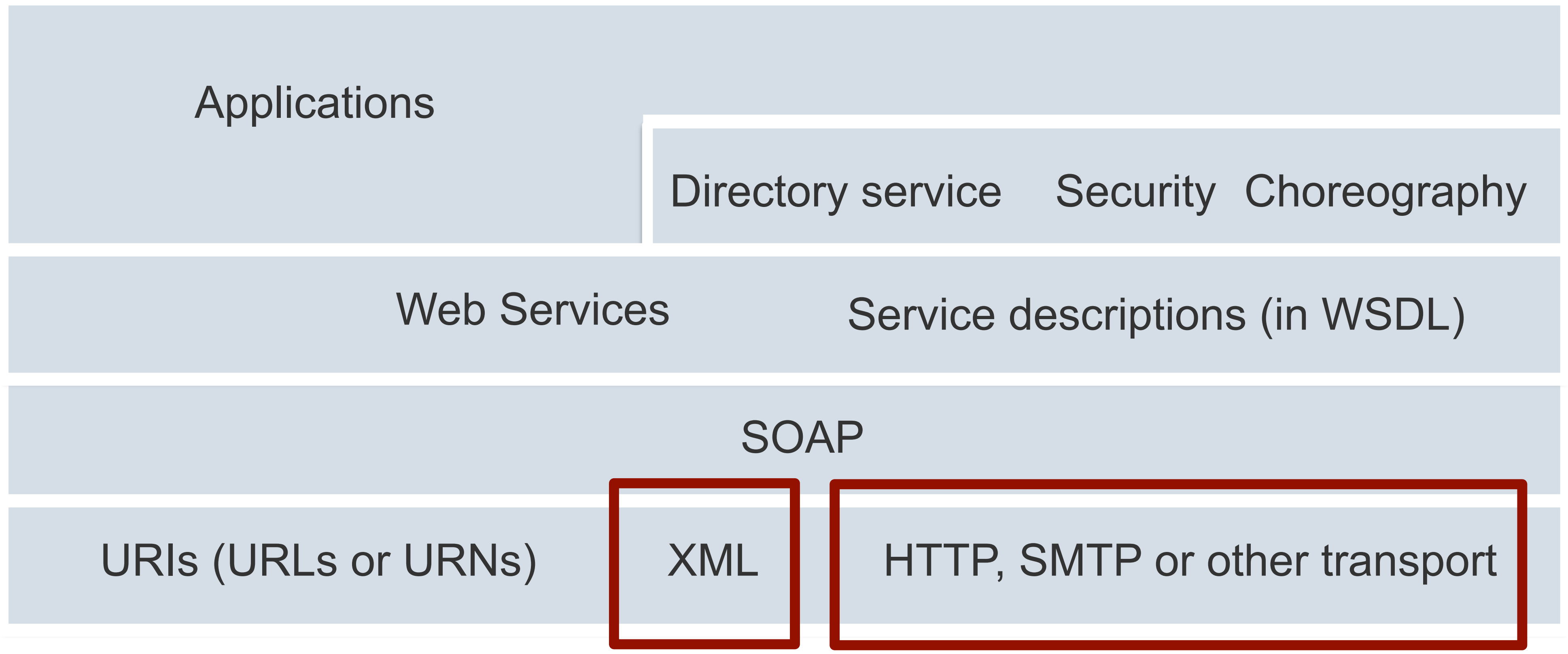
- Abstractions which may not have physical representation
- Requesting a URI yields a *resource representation*
- Should be an appropriate and useful manifestation of the abstraction

Resources can have *different representations*

- In a well-designed environment, you should get what works best for you
- HTML for big screens vs. HTML for mobile devices
- An event calendar based on my location and preferences

(Wilde, 2008)

Web service infrastructure and components

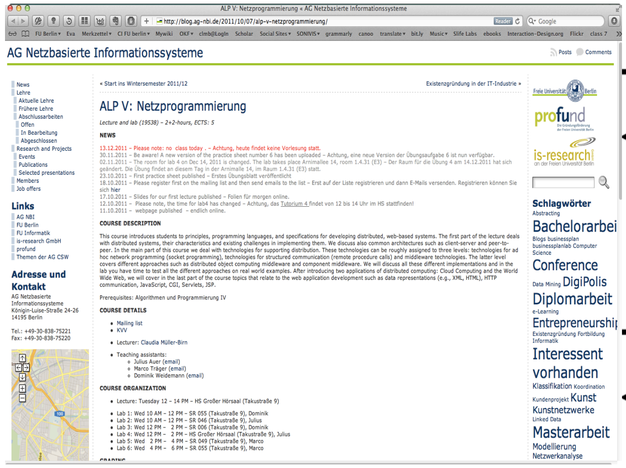


Web Services

The Hypertext Transfer Protocol (HTTP)

DNS & HTTP

The two basic protocols which every Web browser must implement are DNS access and HTTP. However, most operating systems provide an API for DNS access, so the browser can use this service locally and only has to implement HTTP. TCP (which is required as the foundation for HTTP) is usually provided by the operating system.


ALP V: Netzprogrammierung « AG Netzbasierte Informationssysteme

http://blog.ag-nbi.de/2011/10/07/alp-v-netzprogrammierung/

FU Berlin ▾ Eva Merkzettel ▾ CI FU berlin ▾ Mywiki OKF ▾ clmb@LogIn Scholar Social Sites ▾ SONIVIS ▾ grammarly canoo translate ▾ bit.ly Music ▾ Slife Labs ebooks Interaction-Design.org Flickr class 7

AG Netzbasierte Informationssysteme

Posts Comments

- News
- Lehre
  - Aktuelle Lehre
  - Frühere Lehre
  - Abschlussarbeiten
    - Offen
    - In Bearbeitung
    - Abgeschlossen
- Research and Projects
  - Events
  - Publications
  - Selected presentations
- Members
- Job offers

Links

- AG NBI
- FU Berlin
- FU Informatik
- is-research GmbH
- profund
- Themen der AG CSW

Adresse und Kontakt

AG Netzbasierte Informationssysteme
Königin-Luise-Straße 24-26
14195 Berlin

Tel.: +49-30-838-75221
Fax: +49-30-838-75220

« Start ins Wintersemester 2011/12 Existenzgründung in der IT-Industrie »

ALP V: Netzprogrammierung

Lecture and lab (19538) – 2+2-hours, ECTS: 5

NEWS

13.12.2011 – Please note: no class today . – Achtung, heute findet keine Vorlesung statt.
30.11.2011 – Be aware! A new version of the practice sheet number 6 has been uploaded – Achtung, eine neue Version der Übungsaufgabe 6 ist nun verfügbar.
02.11.2011 – The room for lab 4 on Dec 14, 2011 is changed. The lab takes place Arnimallee 14, room 1.4.31 (E3) – Der Raum für die Übung 4 am 14.12.2011 hat sich geändert. Die Übung findet an diesem Tag in der Arnimalle 14, im Raum 1.4.31 (E3) statt.
23.10.2011 – First practice sheet published – Erstes Übungsblatt veröffentlicht
18.10.2011 – Please register first on the mailing list and then send emails to the list – Erst auf der Liste registrieren und dann E-Mails versenden. Registrieren können Sie sich hier
17.10.2011 – Slides for our first lecture published – Folien für morgen online.
12.10.2011 – Please note, the time for lab4 has changed – Achtung, das Tutorium 4 findet von 12 bis 14 Uhr im HS stattfinden!
11.10.2011 – webpage published – endlich online.

COURSE DESCRIPTION

This course introduces students to principles, programming languages, and specifications for developing distributed, web-based systems. The first part of the lecture deals with distributed systems, their characteristics and existing challenges in implementing them. We discuss also common architectures such as client-server and peer-to-peer. In the main part of this course we deal with technologies for supporting distribution. These technologies can be roughly assigned to three levels: technologies for ad hoc network programming (socket programming), technologies for structured communication (remote procedure calls) and middleware technologies. The latter level covers different approaches such as distributed object computing middleware and component middleware. We will discuss all these different implementations and in the lab you have time to test all the different approaches on real world examples. After introducing two applications of distributed computing: Cloud Computing and the World Wide Web, we will cover in the last part of the course topics that relate to the web application development such as data representations (e.g., XML, HTML), HTTP communication, JavaScript, CGI, Servlets, JSP.

Prerequisites: Algorithmen und Programmierung IV

COURSE DETAILS

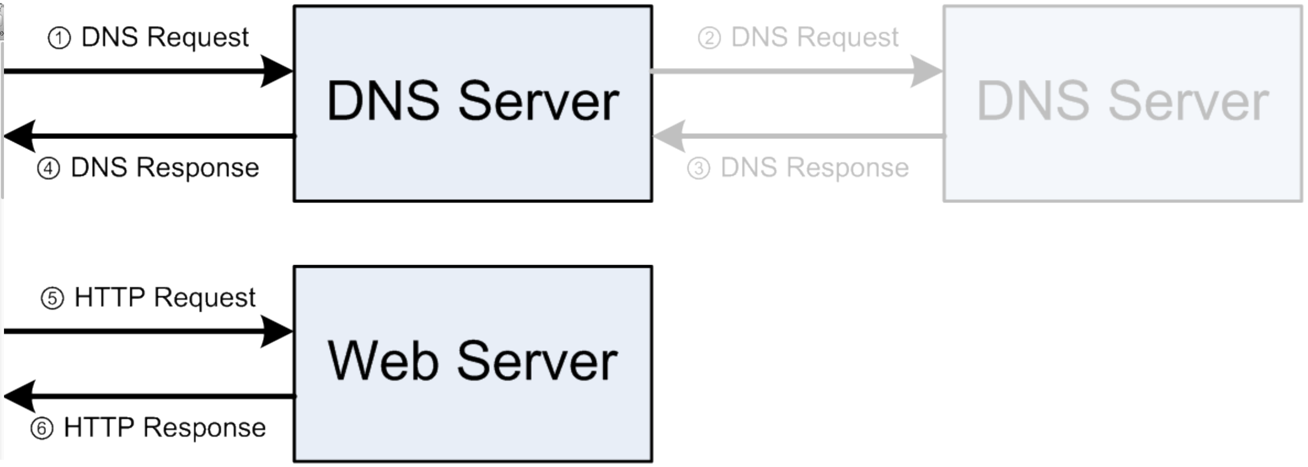
- Mailing list
- KVV
- Lecturer: Claudia Müller-Birn
- Teaching assistants:
  - Julius Auer (email)
  - Marco Träger (email)
  - Dominik Weidemann (email)

COURSE ORGANIZATION

- Lecture: Tuesday 12 – 14 PM – HS Großer Hörsaal (Takustraße 9)
- Lab 1: Wed 10 AM – 12 PM – SR 055 (Takustraße 9), Dominik
- Lab 2: Wed 10 AM – 12 PM – SR 046 (Takustraße 9), Julius
- Lab 3: Wed 12 PM – 2 PM – SR 006 (Takustraße 9), Dominik
- Lab 4: Wed 12 PM – 2 PM – HS Großer Hörsaal (Takustraße 9), Julius
- Lab 5: Wed 2 PM – 4 PM – SR 049 (Takustraße 9), Marco
- Lab 6: Wed 4 PM – 6 PM – SR 055 (Takustraße 9), Marco

Schlagwörter

Abstracting Bachelorarbei Blogs businessplan businessplanlab Computer Science Conference Data Mining DigiPolis Diplomarbeit e-Learning Entrepreneurshi Existenzgründung Fortbildung Informatik Interessent vorhanden Klassifikation Koordination Kundenprojekt Kunst Kunstnetzwerke Linked Data Masterarbeit Modellierung Netzwerkanalyse



(Wilde, 2008)

HTTP messages

HTTP needs a reliable connection

- The foundation for HTTP is the Transmission Control Protocol (TCP)
- DNS resolution yields an IP address
- Open TCP connection to port 80 or port specified in URI (http://rosetta.sims.berkeley.edu:8085/)

HTTP is a text-based protocol

- The connection is used to transmit text messages
- All HTTP messages are human-readable (not all entities, though)
- Basic HTTP operations can be carried out by hand

(Wilde, 2008)

HTTP header fields

Header fields contain information about the message

- *General header:* Date as the message origination date
- *Request header:* Accept-Language indicated language preferences
- *Response header:* Server contains system information
- *Entity header:* Content-Type specifies the media type of the entity

HTTP defines a number of header fields

- Unknown fields must be ignored (extensibility)
- Unstandardized fields should use a “X-” prefix

HTTP is about acting on these fields

- HTTP defines what HTTP implementations must or should do

(Wilde, 2008)

HTTP Requests

After opening a connection, the client sends a request

- The method indicates the action to be performed on the resource
- HTTP's most interesting methods are: GET, HEAD, POST
- Other interesting methods are: PUT, DELETE

The URI identifies the resource to which the request should be applied

- Absolute URIs are required when contacting Proxies
- Absolute paths are required when contacting a server directly
- The URI may contain query information
- Fragment identifiers are not sent (they are interpreted on the client side)

The host header field must be included in every request.

(Wilde, 2008)

HTTP GET

Retrieval action based on the URI

- Maybe implemented by reading a file
- Maybe implemented by processing a file (PHP)
- Maybe implemented by invoking a process

Semantics may change based on header fields

- If-*: only reply with the entity if necessary
- Range: only reply with the requested part of the entity

Cacheability depends on header fields of the response

(Wilde, 2008)

HTTP Responses

The status code is given numerically and as text

- 1xx: Informational - Request received, continuing process
- 2xx: Success - The action was successfully received, understood, and accepted
- 3xx: Redirection - Further action must be taken in order to complete the request
- 4xx: Client Error - The request contains bad syntax or cannot be fulfilled
- 5xx: Server Error - The server failed to fulfill an apparently valid request

Header fields specify additional information

- Information about the server
- Information about the entity (media type, encoding, language)

(Wilde, 2008)

Client Error 4xx

This class of HTTP status message indicates there is a problem or error at the client or user agent end.

Examples

- **400 Bad Request:** The request could not be understood by the server due to malformed syntax.
- **401 Unauthorized:** The request requires user authorization but the authorization codes sent were invalid or the user was not recognized in the system.
- **403 Forbidden:** The server understood the request, but refuses to fulfill it. Authorization, in this case, doesn't matter.
- **404 Not Found:** This is the most easily recognized error message. It states that the URI requested does not exist on the server.

Complete list: http://www.ietf.org/rfc/rfc2616.txt

Server Error 5xx

These error messages are sent when the server is aware that it has had a problem or error.

Examples

- **500 Internal Server Error:** The server encountered something unexpected that didn't allow it to complete the request. This is often seen with CGI scripts that have problems.
- **503 Service Unavailable:** The server is unable to handle the request due to maintenance or a temporary overload of the server.
- **505 HTTP Version Not Supported:** The server does not support the HTTP version that was used to make the request.

Complete list: http://www.ietf.org/rfc/rfc2616.txt

HTTP performance

HTTP/1.0 (RFC1945) allowed one transaction per connection

- TCP connection setup and teardown are expensive
- TCP's *slow start* slows down the initial phase of data transfer
- Typical Web pages use between 10-20 resources (HTML + images)
- Typically, these resources are stored on the same server

HTTP/1.1 (RFC2613) introduces *persistent connections*

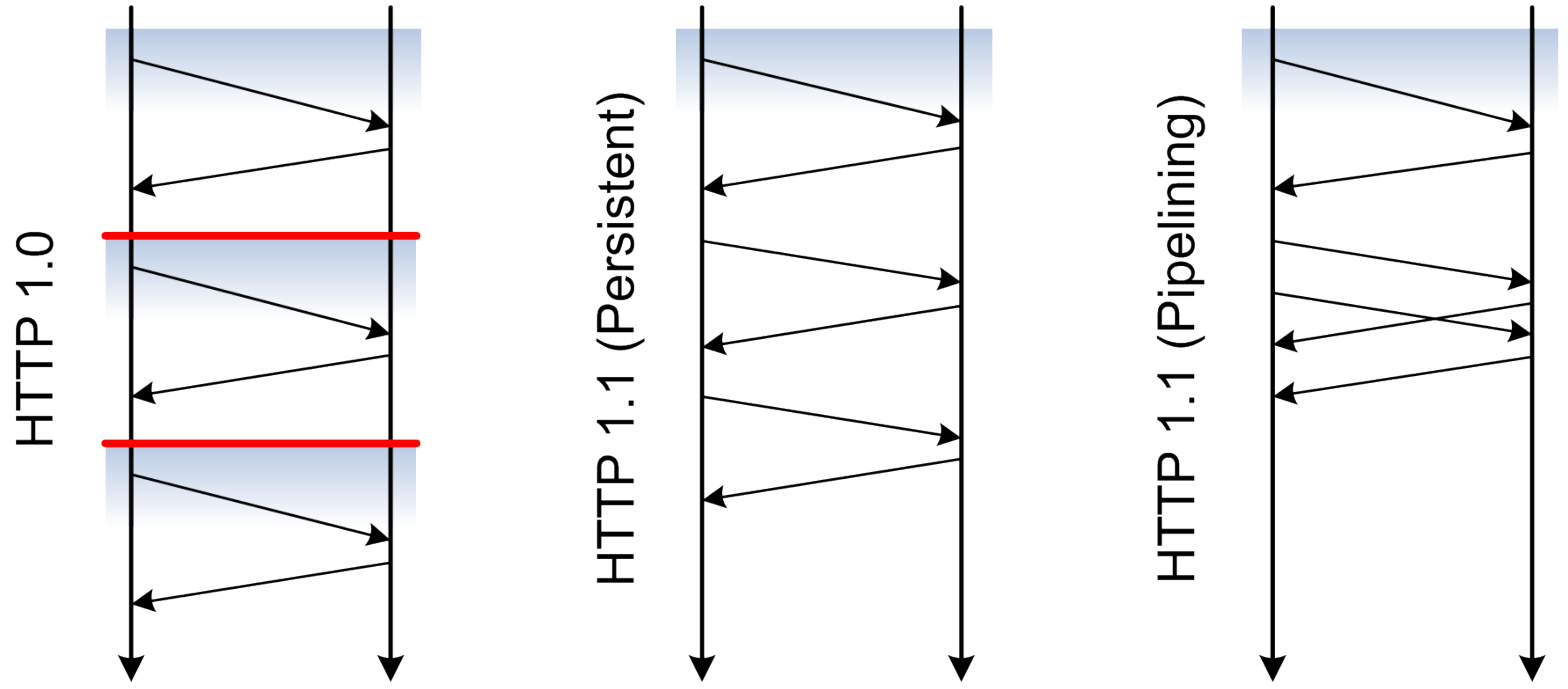
- The TCP connection stays open for some time (10 sec is a popular choice)
- Additional requests to the same server use the same TCP connection

HTTP/1.1 (RFC2613) introduces *pipelined connections*

- Instead of waiting for a response, requests can be queued
- The server responds as fast as possible
- The order may not be changed (there is no sequence number)

(Wilde, 2008)

HTTP connection handling



(Wilde, 2008)

What is content negotiation

Negotiation between two HTTP peers

- Resources may be available in different representations
- Possible dimensions are language, graphics format, character encoding, …
- By using one URI, it should be possible to get the “best” resource

Negotiation requires knowledge about the resource user

- Languages depend on humans reading pages
- Graphics formats depend on the browser's functionality

Content negotiation is a form of a Web-based service

- Client request a URI and have some constraints
- Using these constraints, the best representation should be served
- Ideally, content negotiation should not be too expensive

(Wilde, 2008)

Three different variants

Server Side Content Negotiation

- The server has a set of representations and information from the request
- The server returns the “best” representation based on the request

Client Side Content Negotiation

- The server responds with a list of different representations
- The client (browser or user) makes a choice and sends a second request

Transparent Content Negotiation

- Caches act as in client side negotiation and thus know the available representations
- Clients contacting the cache can be served by the cache as in server side negotiation

(Wilde, 2008)

Web server session management

Session management is used to make the stateless HTTP protocol support session state.

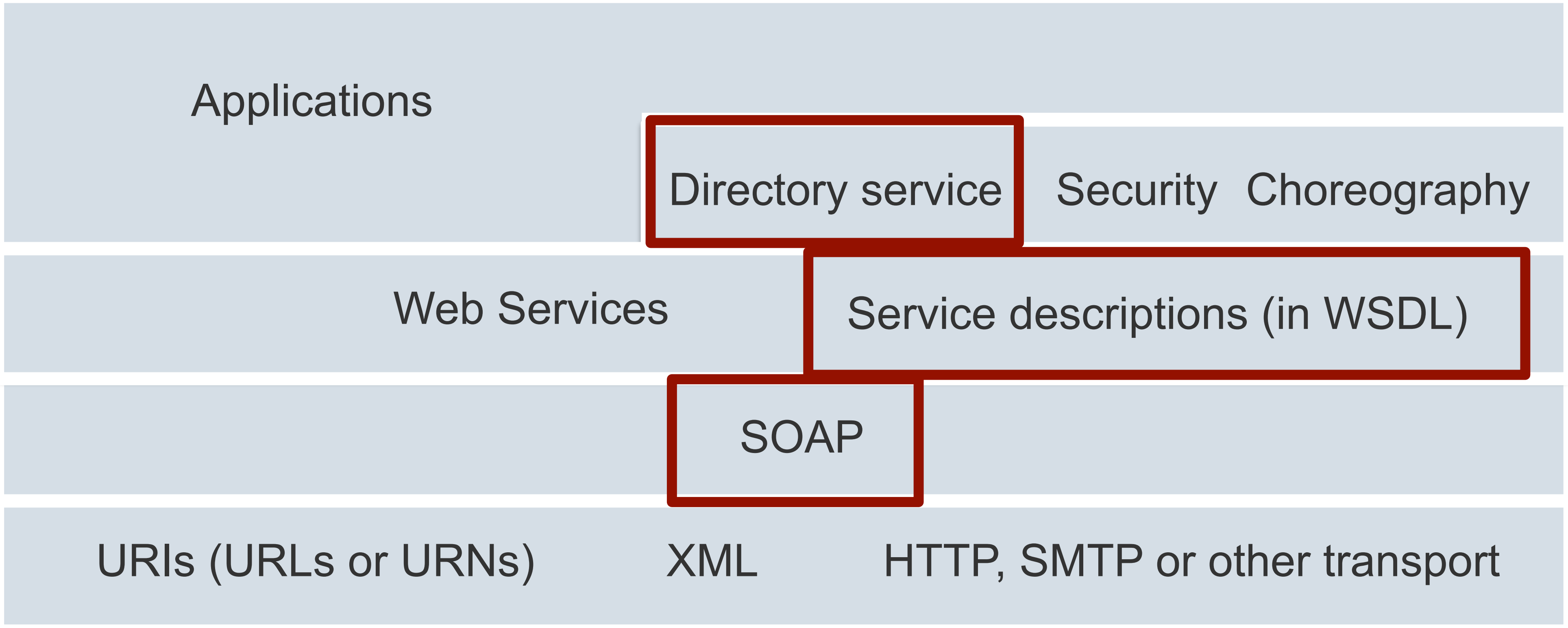
For example, once a user has authenticated oneself to the web server, her next HTTP request (GET or POST) should not cause the web server to ask her for her account and password again.

For this, cookies (RFC 6265) are frequently used. Typically the web server will first send a cookie containing a unique session identifier. Users then submit their credentials and the web application authenticates the session and allows the user access to services.

The session information is stored on the web server using the session identifier (session ID) generated as a result of the first (sometimes the first authenticated) request from the end user running a web browser.

HTTP State Management Mechanism http://tools.ietf.org/html/rfc6265
http://en.wikipedia.org/wiki/Session_management

Web service infrastructure and components



Web Services

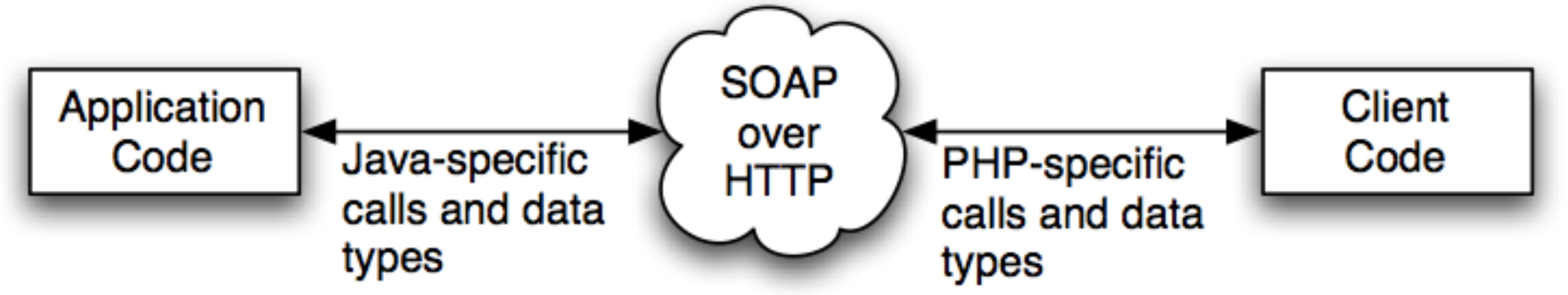
Realizing web services with SOAP

Simple Object Access Protocol (SOAP)

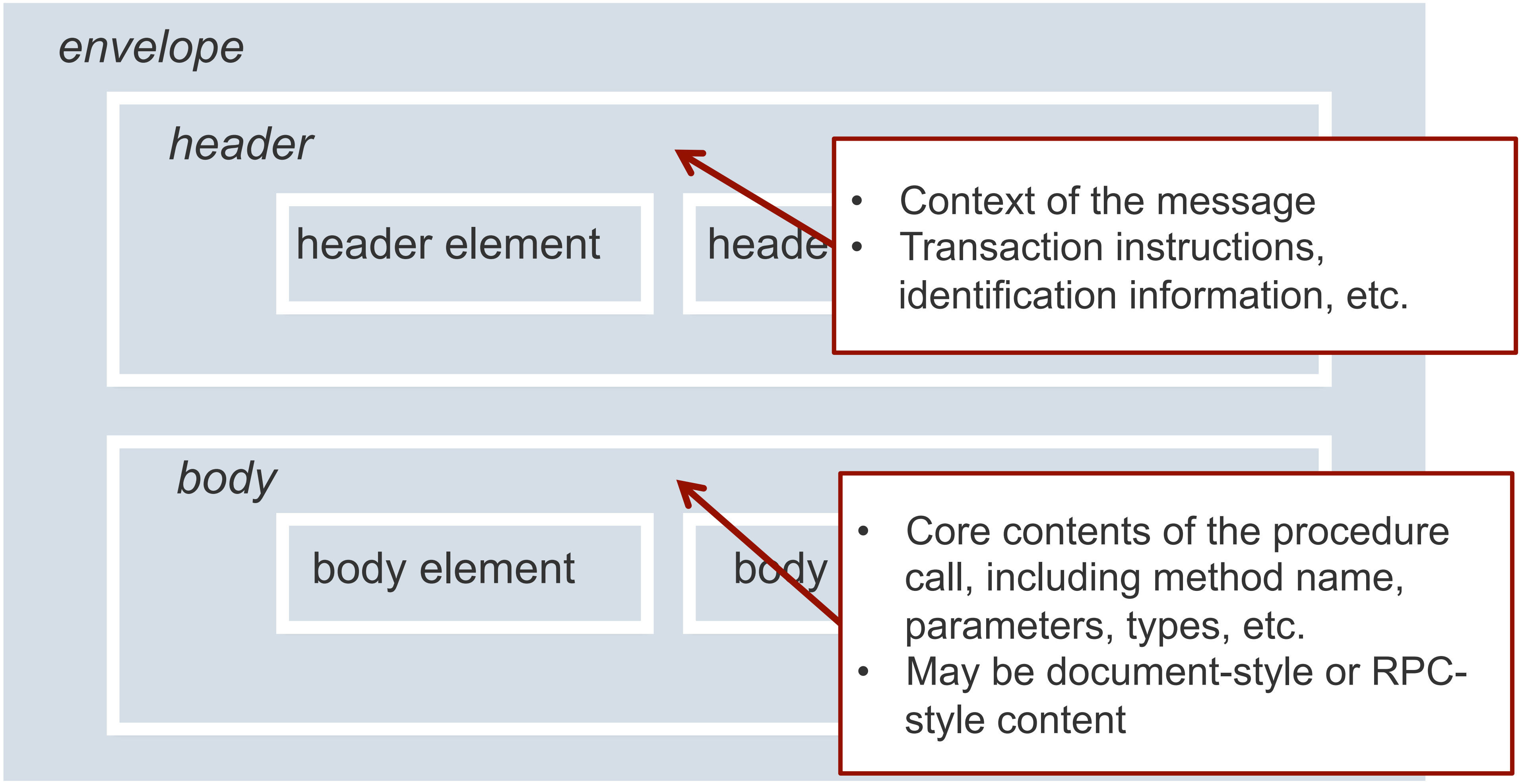
SOAP is designed to enable both client-server and asynchronous interaction over the Internet. It defines a scheme for using XML to represent the contents of request and reply messages as well as a scheme for the communication of documents.

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

(Very) basic SOAP architecture:


Application Code
Java-specific calls and data types
SOAP over HTTP
PHP-specific calls and data types
Client Code

SOAP message in an envelope



- Context of the message
- Transaction instructions, identification information, etc.

- Core contents of the procedure call, including method name, parameters, types, etc.
- May be document-style or RPC-style content

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.

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,

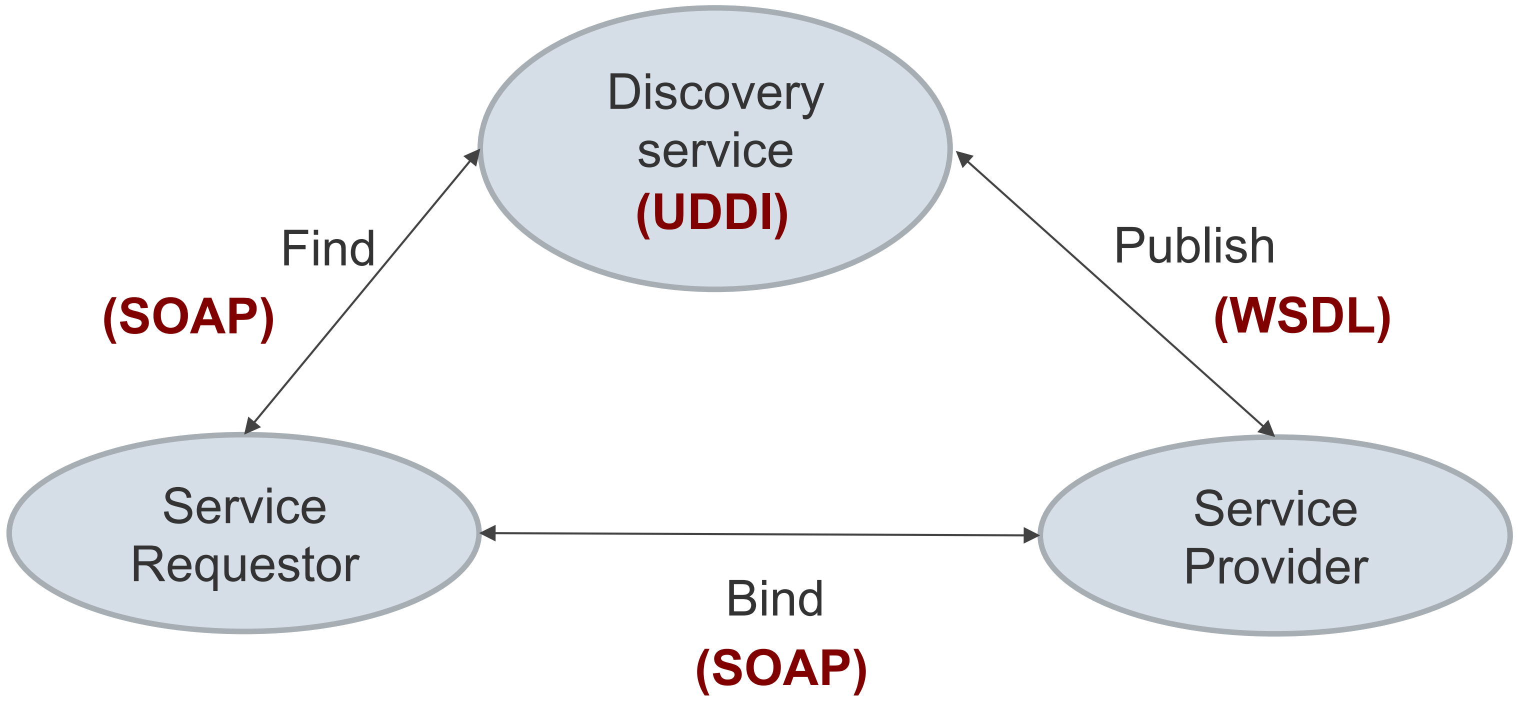
(2) the Web services they make available, and

(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.

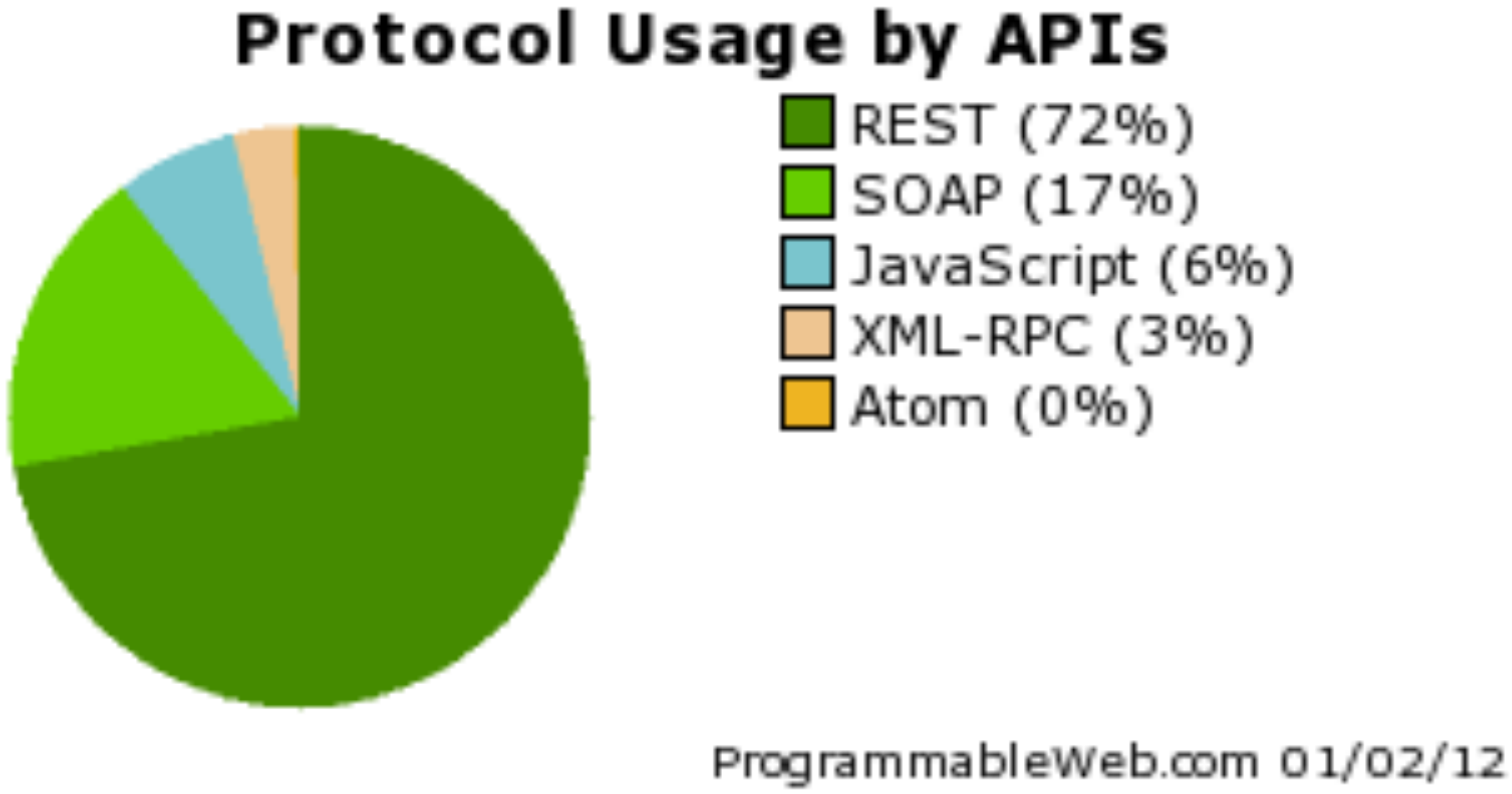
More information at:
http://www.oasis-open.org/committees/uddi-spec

Web service architecture



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

Did we talked about everything?


Protocol Usage by APIs
REST (72%)
SOAP (17%)
JavaScript (6%)
XML-RPC (3%)
Atom (0%)
ProgrammableWeb.com 01/02/12

Service oriented Architecture and Web Services

Representational State Transfer (REST)

What is REST?

REST stands for **Re**presentational **S**tate **T**ransfer 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 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.

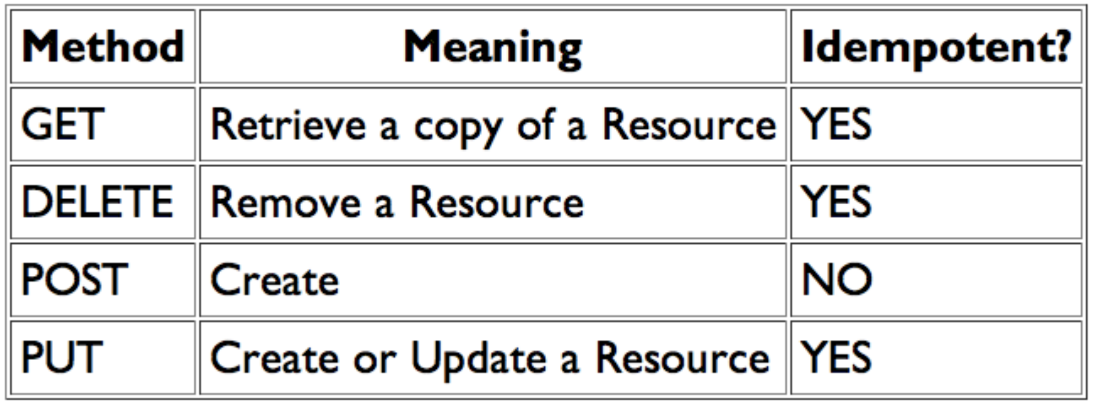
A set of uniquely addressable resources enabled by a universal syntax for resource identification; “Everything is a Resource” in a RESTful system.

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.

REST operations

If we assume the standard four operations from HTTP/1.1 then the “meanings” of these operations would be something like:



Method	Meaning	Idempotent?
GET	Retrieve a copy of a Resource	YES
DELETE	Remove a Resource	YES
POST	Create	NO
PUT	Create or Update a Resource	YES

Stateless Interactions

For many RESTful applications, state is an essential part but the idea of REST is to avoid long-lasting transactions *in applications.*

Statelessness in this context means to move state to clients or resources and the most important consequence is to avoid state in server-side applications.

Resource state is managed on the server. It is the same for every client working with the service and when a client changes resource state other clients see this change as well.

Client state is managed on the client and it is specific for a client and thus has to be maintained by each client. It may affect *access* to server resources, but not the resources themselves.

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.

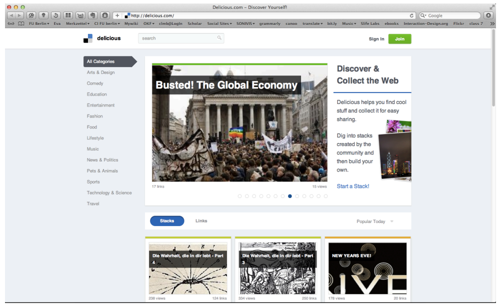
http://www.xfront.com/REST-Web-Services.html

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

Getting Bookmarks

In the POX Delicious API bookmarks are accessed by a RPC style request to

http://del.icio.us/api/posts/get

with a number of optional query string parameters that influence the returned results.

To do a similar thing RESTfully, we'll define a similar resource at

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

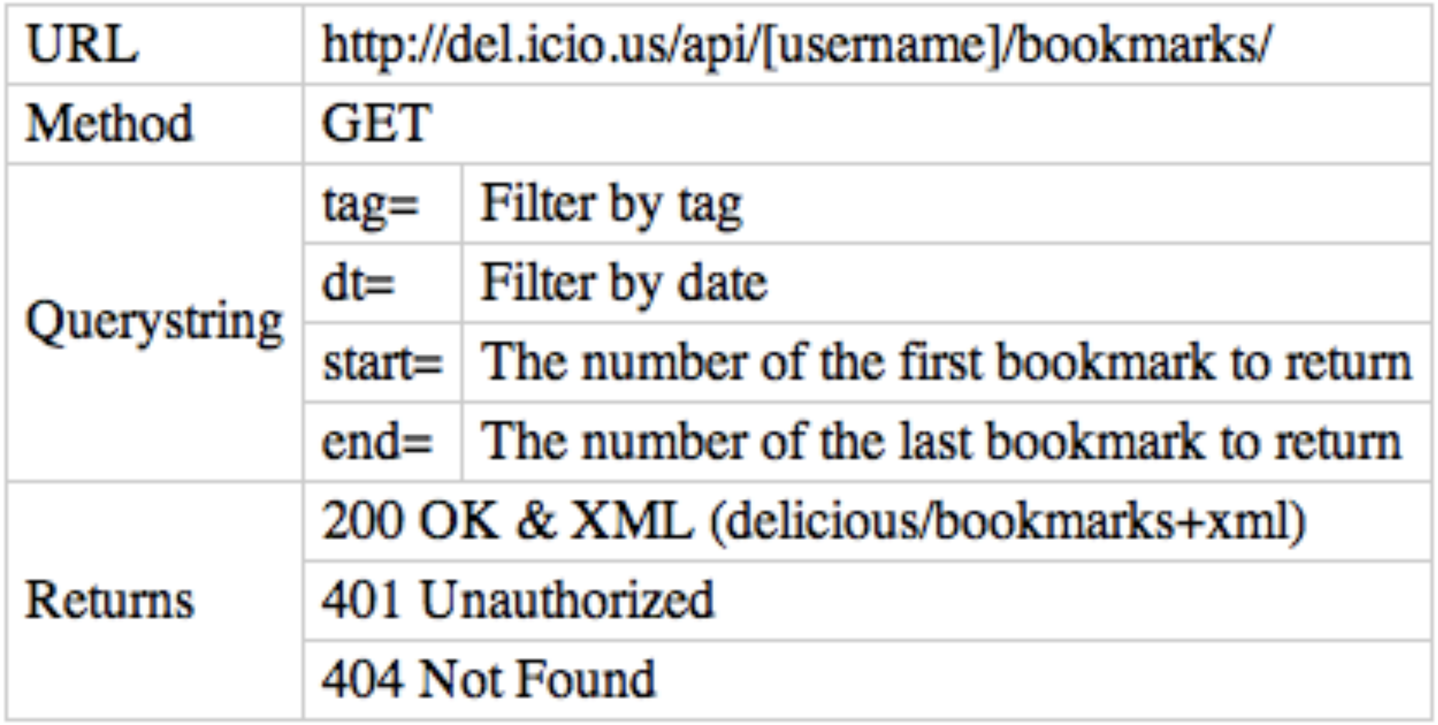
that returns a list of bookmarks.

We'll also define an infinite number of resources at

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

that represent our individual bookmarks where [hash] is the Delicious hash used to identify a bookmark.

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

So, what have we learned today?

- We know now the difference between URIs, URLs and URNs.
- By providing a resource, we should always keep in mind that different representations of resources might be useful.
- You know about the possible HTTP responses.
- There are different approaches to realize web services. Even though, people talk about SOAP, REST is much more applied in practice.
- Difference between SOAP- and REST-based 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