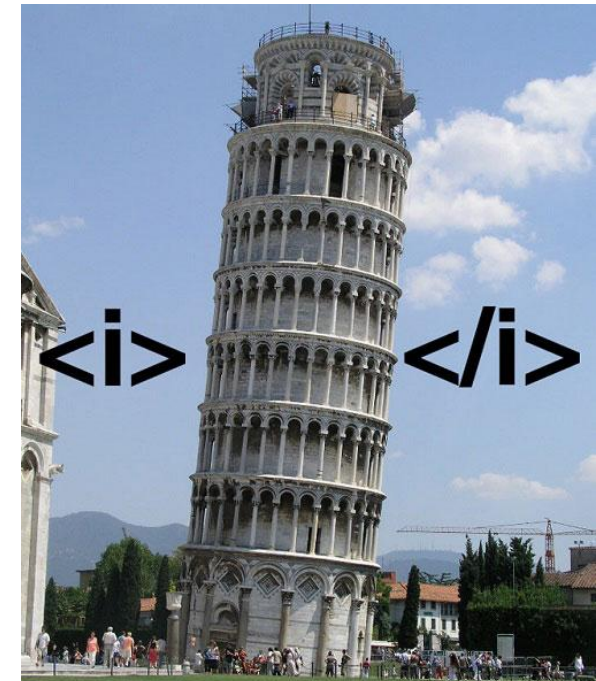




JACOBS
UNIVERSITY

Web Service Protocols

Instructor: Peter Baumann
email: p.baumann@jacobs-university.de
tel: -3178
office: room 60, Research 1



Overview

- HTTP
- SOAP
- REST
- AJAX



JACOBS
UNIVERSITY

HTTP: GET, POST, & Friends

GET Requests

- Recall: http offers
 - GET, POST, PUT, DELETE
 - ...plus several more
- Request modification through key/value pairs
 - ?
 - &
- Client sends:

see REST later!

`http://acme.com/srv ? mybasket=6570616275 & article=656e44204456`

Request Parameters: How Passed?

■ GET parameters: URL text

- Can be cached, bookmarked
- Reload / back in history harmless
- Data visible in URL

GET srv?**k1=v1&k2=v2** HTTP/1.1

■ POST parameters: HTTP message body

- Not cached, bookmarked
- Reload / back in history **re-submits**
- Data not visible,
not in history,
not in server logs

POST srv HTTP/1.1
k1=v1&k2=v2

http://www.w3schools.com/tags/ref_httpmethods.asp



JACOBS
UNIVERSITY

SOAP

XML, SOAP, WSDL, UDDI

- Web Services **four main technologies** (bottom up):
- **XML** (Extensible Markup Language)
 - Encode & organize the Message
- **SOAP** (Simple Object Access Protocol)
 - Defines message standards and acts as message envelope
- **WSDL** (Web Service Description Language)
 - Describes a web service and its functions
- **UDDI** (Universal Description, Discovery and Integration Service)
 - Dynamically find other web services

What is SOAP?

- Used to stand for **Simple Object Access Protocol**
 - but it is no longer an acronym
- SOAP is a protocol which allows ...
 - **exchanging** structured and typed information **between peers** in a decentralized and distributed environment
 - **accessing** services, objects and servers in a platform-independent manner
- Encompasses: Envelope + encoding rules + RPC
 - XML
- Main Goal:
 - Facilitate **interoperability** across platforms and programming languages

*Operations –
that's what was
missing with XML*

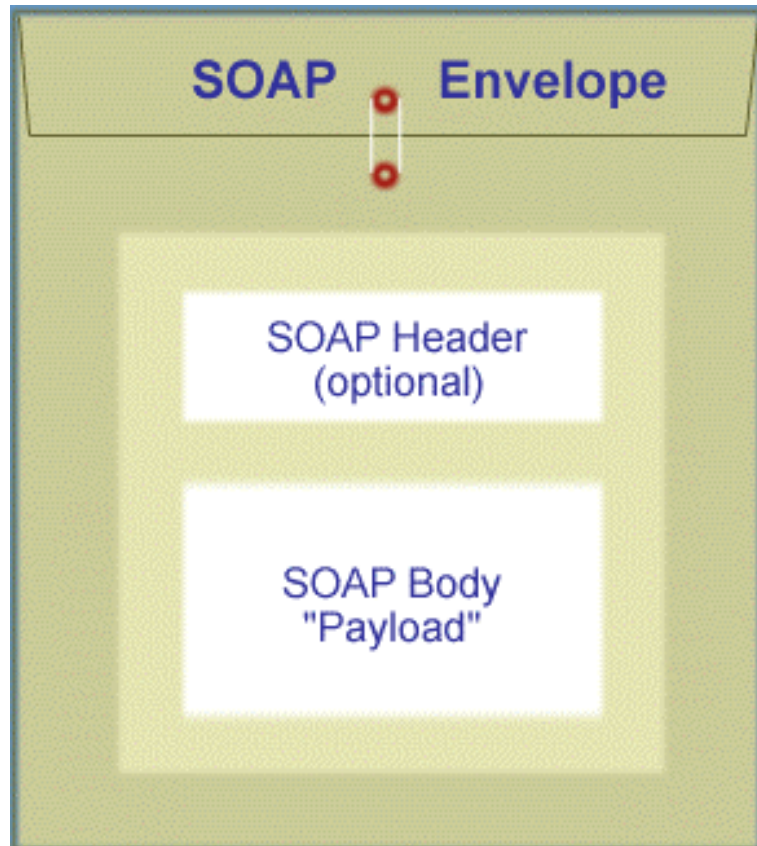
Example

- Google API SOAP 1.1 msg
 - Searching for “boston”, “university”

```
<?xml version='1.0' encoding='UTF-8'?>
<soap11:Envelope xmlns="urn:GoogleSearch"
  xmlns:soap11="http://schemas.xmlsoap.org/soap/envelope/">
  <soap11:Body>
    <doGoogleSearch>
      <key>000000000000000000000000000000000000</key>
      <q>boston university</q>
      <start>0</start>
      <maxResults>10</maxResults>
      <filter>true</filter>
      <restrict></restrict>
      <safeSearch>false</safeSearch>
      <lr></lr>
      <ie>latin1</ie>
      <oe>latin1</oe>
    </doGoogleSearch>
  </soap11:Body>
</soap11:Envelope>
```

SOAP Message Structure

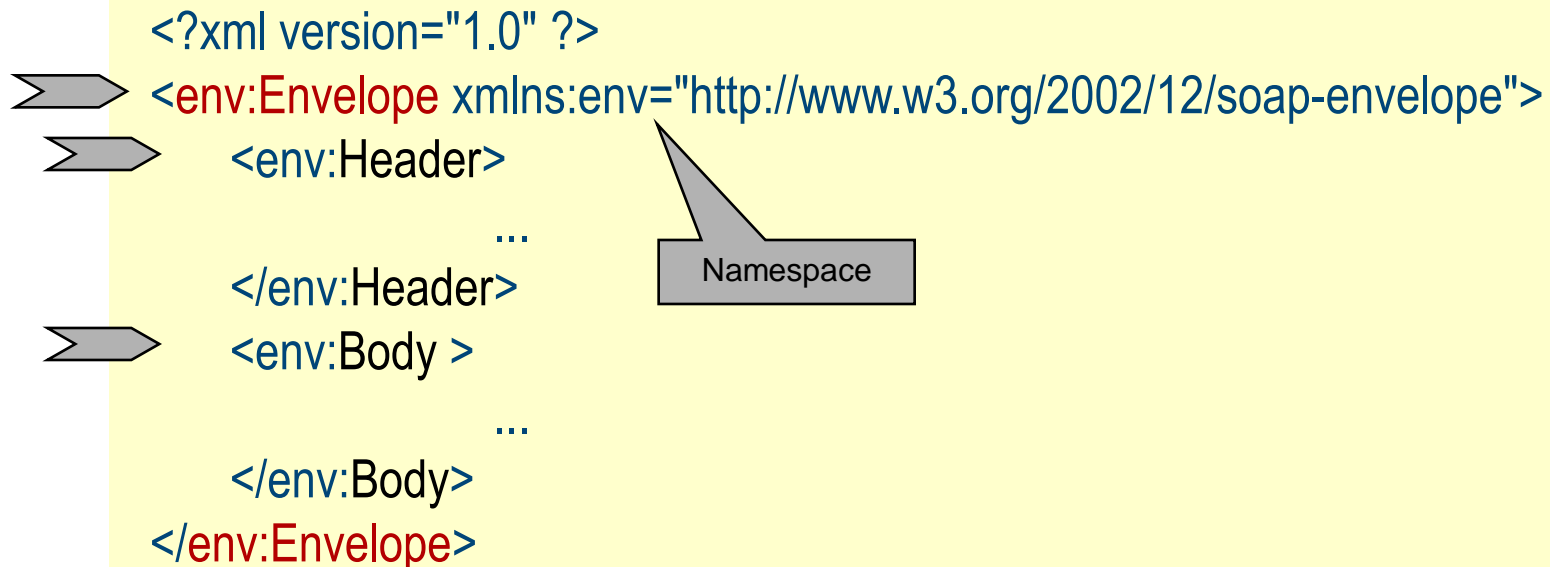
- SOAP Envelope
 - Required
- SOAP Header
 - Optional
- SOAP Body
 - Required



SOAP Envelope

- Root of a SOAP Message
- Contains a SOAP Header (optional) and a SOAP Body
- Example:

```
<?xml version="1.0" ?>  
➤ <env:Envelope xmlns:env="http://www.w3.org/2002/12/soap-envelope">  
  ➤ <env:Header>  
    ...  
  </env:Header>  
  ➤ <env:Body >  
    ...  
  </env:Body>  
</env:Envelope>
```



The diagram illustrates the structure of a SOAP Envelope. It shows the XML code with annotations. Three chevron arrows point to the opening tags of `<env:Envelope>`, `<env:Header>`, and `<env:Body>`. A callout box labeled "Namespace" points to the `xmlns:env` attribute in the `<env:Envelope>` tag.

SOAP Header: Example

Namespace

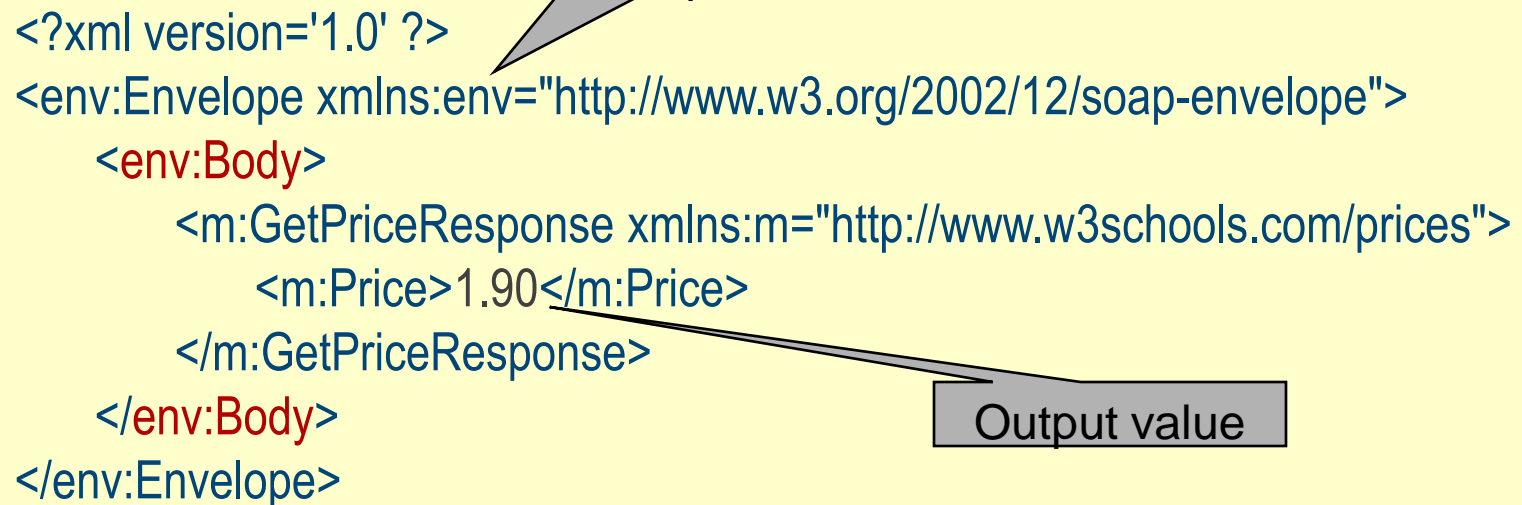
```
<?xml version='1.0' ?>
<env:Envelope xmlns:env="http://www.w3.org/2002/12/soap-envelope">
  <env:Header>
    <m:reservation xmlns:m="http://travelcompany.example.org/reservation"
      env:role="http://www.w3.org/2002/12/soap-envelope/role/next"
      env:mustUnderstand="true">
      ...
    </m:reservation>
    ...
  </env:Header>
  <env:Body>
    ...
  </env:Body>
</env:Envelope>
```

e.g. Context information:

- ...*role/next*: intermediary, ultimate receiver
- ...*role/none*: nodes must not act in this role
- ...*role/ultimateReceiver*: to act as recipient

SOAP Body

- Mandatory
- Contains (application specific) information to the recipient + SOAP Fault
- Example:



The diagram shows an XML snippet for a SOAP body. A yellow background highlights the XML code. A grey box labeled "Namespace" has a pointer to the `xmlns:env` attribute in the `<env:Envelope>` tag. Another grey box labeled "Output value" has a pointer to the `1.90` value inside the `<m:Price>` tag.

```
<?xml version='1.0' ?>
<env:Envelope xmlns:env="http://www.w3.org/2002/12/soap-envelope">
  <env:Body>
    <m:GetPriceResponse xmlns:m="http://www.w3schools.com/prices">
      <m:Price>1.90</m:Price>
    </m:GetPriceResponse>
  </env:Body>
</env:Envelope>
```

who defines body syntax?

SOAP Fault

- For error handling within a SOAP application

- Example:

```
<?xml version='1.0' ?>
<env:Envelope xmlns:env="http://www.w3.org/2002/12/soap-envelope">
  <env:Body>
    <env:Fault>
      <env:Code>
        <env:Value>env:MustUnderstand</env:Value>
      </env:Code>
      <env:Reason>
        <env:Text xml:lang="en-US">Header not understood</env:Text>
        <env:Text xml:lang="fr">En-tête non compris</env:Text>
      </env:Reason>
    </env:Fault>
  </env:Body>
</env:Envelope>
```

Namespace

mandatory

SOAP Fault code

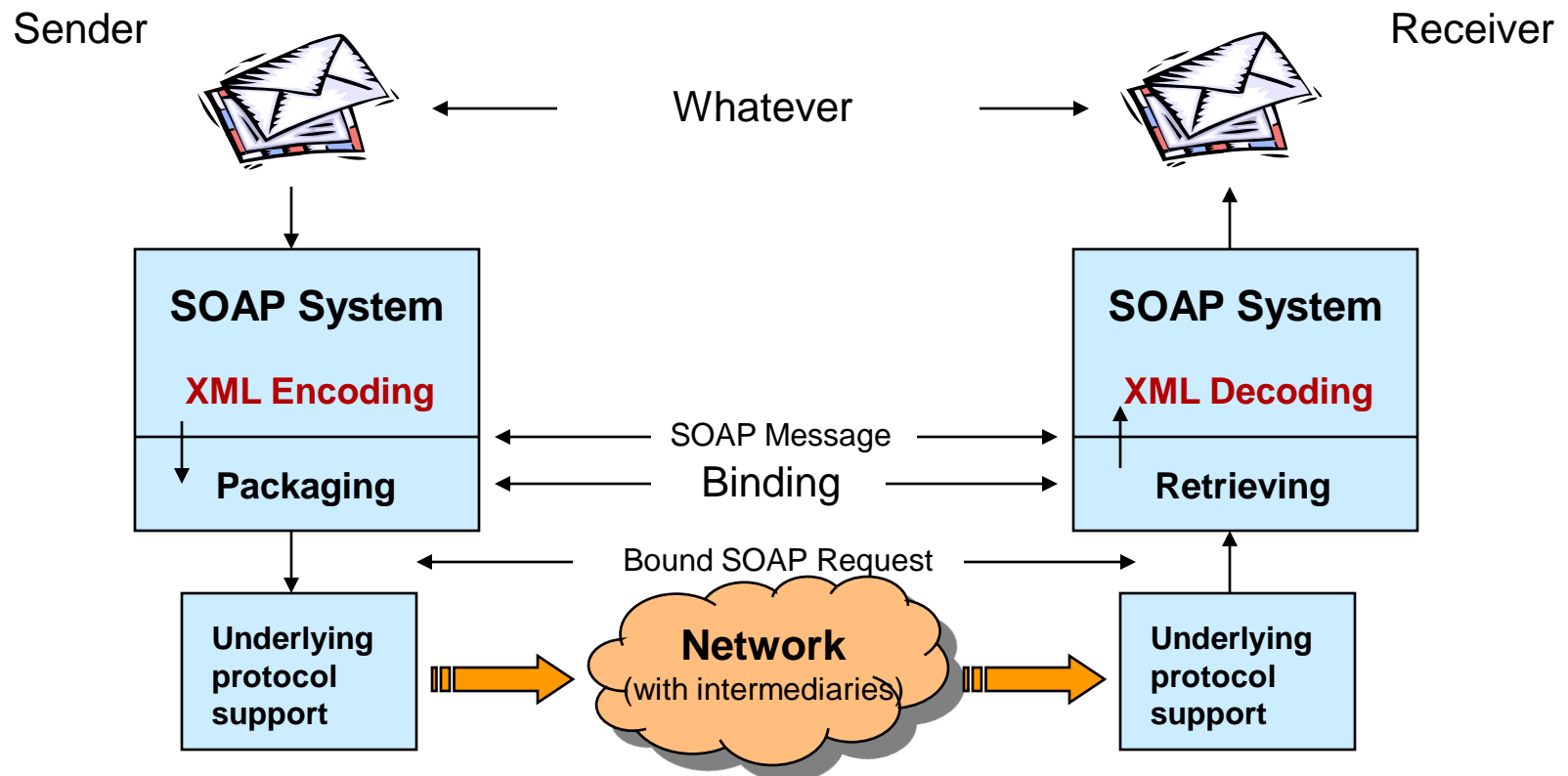
mandatory

Human readable explanation of fault (here in different languages)

SOAP Envelope: XML Schema

```
- <xs:schema targetNamespace="http://schemas.xmlsoap.org/soap/envelope/">
  <!-- Envelope, header and body -->
  <xs:element name="Envelope" type="tns:Envelope"/>
  <xs:complexType name="Envelope">
    <xs:sequence>
      <xs:element ref="tns:Header" minOccurs="0"/>
      <xs:element ref="tns:Body" minOccurs="1"/>
      <xs:any namespace="##other" minOccurs="0" maxOccurs="unbounded" processContents="lax"/>
    </xs:sequence>
    <xs:anyAttribute namespace="##other" processContents="lax"/>
  </xs:complexType>
  <xs:element name="Header" type="tns:Header"/>
  <xs:complexType name="Header">
    <xs:sequence>
      <xs:any namespace="##other" minOccurs="0" maxOccurs="unbounded" processContents="lax"/>
    </xs:sequence>
    <xs:anyAttribute namespace="##other" processContents="lax"/>
  </xs:complexType>
  <xs:element name="Body" type="tns:Body"/>
  <xs:complexType name="Body">
    <xs:sequence>
      <xs:any namespace="##any" minOccurs="0" maxOccurs="unbounded" processContents="lax"/>
    </xs:sequence>
    <xs:anyAttribute namespace="##any" processContents="lax">
      <xs:annotation>
        <xs:documentation>
          Prose in the spec does not specify that attributes are allowed on the Body element
        </xs:documentation>
      </xs:annotation>
    </xs:anyAttribute>
  </xs:complexType>
```

SOAP Architecture



Ex: Google API: Java on SOAP

```
import com.google.soap.search.*;
public class Test
{
    public static void main(String[] args)
    {
        try
        {
            GoogleSearch search = new GoogleSearch();
            search.setQueryString(args[0]);
            GoogleSearchResult result = search.doSearch();
            System.out.println(result.toString());
        }
        catch(Exception e)
        {
            e.printStackTrace();
        }
    }
}
```

www.google.com/apis

Wrap-Up: Pros & Cons of SOAP

- SOAP = HTTP + XML for Web Service messaging with **server-side code invocation**

- Advantages:

- Interoperability
- Extensibility
- Vendor-neutral
- Independent of platforms and programming languages
- Firewall-friendly (?)

- Disadvantages:

- Lack of **security**
...custom security measures on top of SOAP → loss of interoperability
- Lack of **efficiency**
...most time used in en-/decoding

- Powerful, but inherently **dangerous**



JACOBS
UNIVERSITY

REST

(Representational State Transfer)

Ranting Against SOAP

- SOAP \neq remote function invocation
 - does not really hide underlying message passing principle
- SOAP defines only syntax, not semantics of operations
 - API = fct name + parameters
- Quite complex for non-programmers who "just want a Web service"
- *...anything else out there beyond SOAP and XML-RPC?*

REST

[Thomas Roy Fielding, 2002]

- REST
= **Representational State Transfer**
 - Resource + URI
 - *Web = one address space*
 - representation
 - Client requests follow xlink
 - *→ new state*
- Not a standard nor product, but „**architectural style**“
 - = way to craft Web interface
- URI defines resource being requested
 - Consistent design philosophy
 - easy to follow
- Relies on four basic http operations:
 - GET – *Query*
 - POST – *Update*
 - PUT – *Add*
 - DELETE – *Delete*

Sample RESTful Application

- *Scenario: online shop*
- Fetch information: "shopping basket with id 5873" GET /shoppingBasket/5873
 - Response:

```
<shoppingBasket xmlns:xlink="http://www.w3.org/1999/xlink">
  <customer xlink:href="http://shop.oio.de/customer/5873">5873</customer>
  <position nr="1" amount="5">
    <article xlink:href="http://shop.oio.de/article/4501" nr="4501">
      <description>lollypop</description>
    </article>
  </position>
  <position nr="2" amount="2">... </position>
</shoppingBasket>
```
 - Client can follow links, that changes its state
 - No side effect (status change) on server side

Sample RESTful Application (contd.)

■ Place order:

"add article #961 to shopping basket #5873"

- Changes server state

```
POST /shoppingBasket/5873
articleNr=961
```

■ Add article

- Again, changes server state
- Returns new id

```
PUT /article
```

```
<article>
  <description>Rooibush tea</description>
  <price>2.80</price>
  ...
</article>
```

```
HTTP/1.1 201 OK
```

```
...
```

```
http://shop.oio.de/article/6005
```

■ Delete article

- Server state change

```
DELETE /article/6005
```

Choice of Return Formats

- Problem: how to indicate output format
 - Ex: Old browsers understood GIF, JPEG for imagery
 - GET/KVP: `http://.../service-endpoint?q=...&format=image/tiff`
- REST: use http **Accept-Encoding** parameter [IETF RFC 2616]
 - More powerful than GET: negotiate **alternatives**, **quality factor** $q \in [0..1]$
 - However, RESTafarians typically ignore this, use „...&f=...“ ...back to GET/KVP ;-)
- Examples:
 - Accept-Encoding: **compress, gzip**
 - Accept-Encoding:
 - Accept-Encoding: *****
 - Accept-Encoding: **compress;q=0.5, gzip;q=1.0**
 - Accept-Encoding: **gzip;q=1.0, identity; q=0.5, *;q=0**

Security

- Remember: SOAP, XML-RPC do http tunneling
 - Major security leak:
cannot determine request payload unless body is inspected and understood (!)
- REST: typed requests, firewall can judge → better security

```
hermes.oio.de - - [26/Nov/2002:12:43:07 +0100] "GET /shoppingBasket/6 HTTP/1.1" 200
hermes.oio.de - - [26/Nov/2002:12:43:08 +0100] "GET /article/12 HTTP/1.1" 200
hermes.oio.de - - [26/Nov/2002:12:43:08 +0100] "GET /article/5 HTTP/1.1" 200
hermes.oio.de - - [26/Nov/2002:12:43:09 +0100] "POST /shoppingBasket/6 HTTP/1.1" 200
hermes.oio.de - - [26/Nov/2002:12:43:13 +0100] "POST /shoppingBasket/6 HTTP/1.1" 200
hermes.oio.de - - [26/Nov/2002:12:43:14 +0100] "GET /Order/3 HTTP/1.1" 200
```

- → *admins much more inclined to open firewall for REST services than for SOAP*

REST: Appraisal

■ Strengths

- Simple paradigm; Web = RESTful resource *(SOAP: individual spec per service)*
- Caching supported *(SOAP: based on POST, not cached)*
- Proven base stds: http, URI, MIME, XML *(SOAP: WSDL, UDDI, WS-*, BPEL, ...)*
- Scalability:
stateless → resources independent; MIME for new formats; independent deployment;
service composition ("orchestration")
 - *Oops: cookies break REST paradigm*
- *Legacy service integration ("webifying")*

REST: Appraisal - Weaknesses

- Assumes addressability by path + identifier (URI!) = single-root hierarchies (XML centric)
 - no complex queries: only conjunctive queries, no nesting, no ...

`http://acme.com/endpoint?q=select%20...%20from%20A,B,C%20where%20...`
- Schema to represent all URIs is complex
- Response data structure definition outside REST (how was that with SOAP?)
- limited support for HTTP PUT & DELETE in popular development platforms

REST: Appraisal (contd.)

- Who uses REST?
 - WebDAV, blogosphere, Atom Publishing Protocol, Ruby on Rails
 - Hot discussion topic in OGC
 - Amazon, Google, Meerkat (O'Reilly)
- Tool support
 - *Tools? What tools?* Apache, IIS, Tomcat, ...

SOAP vs REST

■ SOAP

- Explicit protocol definition, specific services
- ...hence streamlining possible
- Security issues
- *More suitable for bespoke heavy-weight apps*

■ REST

- Plain old http – *"there is no spoon"*
- Transports complete representations of resources, can be less efficient than CORBA, RMI, DCOM, ...
- REST architecture originally designed for massive scale hypermedia distribution
- *More suitable for simple mass apps with unknown #users, #objects*

Selected REST Resources

- Thomas Roy Fielding: Architectural Styles and the Design of Network-based Software Architectures
 - <http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm>
- Second Generation Web Services
 - <http://www.xml.com/pub/a/2002/02/06/rest.html>
- Rest Wiki
 - <http://internet.conveyor.com/RESTwiki/moin.cgi>
- Prescod, Paul: REST and the Real World
 - <http://www.xml.com/lpt/a/2002/02/20/rest.html>
- Prescod, Paul: The Emperor's New Tags - The SOAP/REST Controversy
 - http://www.prescod.net/rest/soap_rest_short.ppt

Summary

- Web services: want function invocation on server
→ Remote Procedure Call (RPC)
 - Existing since 1980s: XDR
 - Web: SOAP
- Web World is evolving
 - New paradigms emerging (and some disappearing)
 - GET/KVP, POST/XML, SOAP, REST, JSON
- Service protocol independent from database query languages!
 - Ex: `http://acme.com/access-point?q=select%20*%20from...`
`<query>select *from...</query>`



JACOBS
UNIVERSITY

AJAX

(Asynchronous Javascript and XML)

History

- Challenge: want **more interactivity** than "click link / reload *complete* page"
 - HTML's `iframes`
- Microsoft IE5 XMLHttpRequest object
 - Outlook Web Access, supplied with Exchange Server 2000
- 2005: term "AJAX" coined by Jesse James Garnett
- made popular in 2005 by Google Suggest
 - start typing into Google's search box → list of suggestions

- AJAX = Asynchronous Javascript and XML
- web development technique for creating more interactive web applications
 - Goal: increase interactivity, speed, functionality, usability
 - not complete page reload → small data loads → more responsive
- asynchronous: c/s communication independent from normal page loading
 - JavaScript
 - XML
 - any server-side PL


Constituent Technologies

- The core: JavaScript **XMLHttpRequest** object
 - Sends data, waits for response via event handler
 - Replaces <FORM> and HTTP GET / POST
- **Client DOM manipulated** to dynamically display & interact
 - Inject response into any place(s) of DOM tree
 - client-side scripting language: JavaScript, Jscript, ...
- Some data format
 - XML, JSON, HTML, text, ...
- Some server agent
 - Servlet, script, ...

Ajax Example: Traditional Style

- Client:

```
<form method='GET' action='http://.../ajax-ex.php'>
  word:
  <input name='wordKey' type='text'>
  <input type='submit' value='Go'>
</form>
```



word:

- Server:

```
<?
  echo 'You have entered ' . $_GET['wordKey']
    . ' and your IP is: ' . $_SERVER['REMOTE_ADDR'];
?>
```

- Client, after **page reload**: You have entered Moribundus, and your IP is:
127.0.0.1

Step 1: Avoid Complete Page Reload

```
<form name='wordForm'>
  word:
  <input name='wordKey' type='text'>
  <input type='button' value='Go' onClick='JavaScript:callBack()'>
  <div id='result'></div>
</form>
```

```
function callBack()
{
  var SERVICE = 'http://.../ajax-ex.php';
  var req = new XMLHttpRequest();
  var val = document.forms['wordForm'].wordKey.value;
  req.open( 'GET', SERVICE+'?wordKey='+val, true );
  req.setRequestHeader( 'Content-Type',
                        'application/x-www-form-urlencoded' );
  req.send( null );
  req.onreadystatechange = function()
  {
    if (req.readyState == 4)
      document.forms['wordForm'].result.innerHTML =
        req.responseText;
  }
}
```

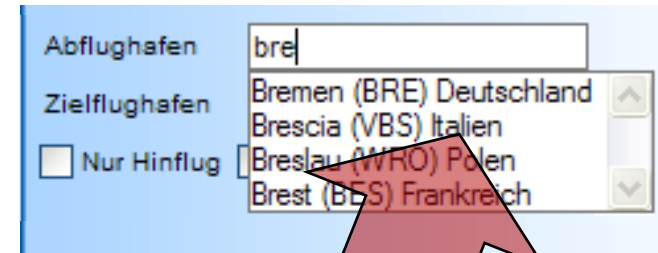
- 0 request not initialized
- 1 request set up
- 2 request sent
- 3 request in process
- 4 request complete

word: _____

You have entered Moribundus, and your IP is: 127.0.0.1

Step 2: Avoid SUBMIT Button

- Before: just re-implemented submit; now: allow c/s activity at **any time**
 - Event handlers
- Ex: suggest keywords with every char typed
 - No submit button!



```
<input name='wordKey' onKeyUp='JavaScript:callBack()'>
```

```
<? ...  
$query = "select entry from Airports  
         where entry like '" . $_GET['wordKey'] . "%'";  
$result = mysql_query( $query );  
while ( $row = mysql_fetch_array( $result ) )  
{  
    print $row[ 'entry' ] . ",";  
}  
?>
```

*How to ship back
& inject data?*

JSON

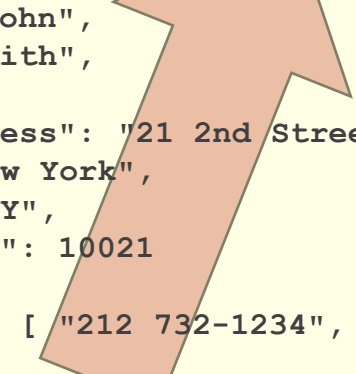
- JSON = JavaScript Object Notation
 - Lightweight data interchange format
 - MIME type: `application/json` (RFC 4627)
 - text-based, human-readable
- alternative to XML use
 - Subset of JavaScript's object literal notation
 - 10x faster than XML parsing
 - `_way_` easier to handle
 - JSON parsing / generating code readily available for many languages

"JSON is XML without garbage"

JSON Example

- Server sends:
- JSON string sent from server:
- response parsing code:

```
req.onreadystatechange=function()  
{  if(req.readyState==4)  
    {  var p = eval( "(" + req.responseText + ")" );  
      document.myForm.firstName.value = p.firstName;  
    }  
}
```



```
{  "firstName": "John",  
    "lastName": "Smith",  
    "address":  
    {  "streetAddress": "21 2nd Street",  
        "city": "New York",  
        "state": "NY",  
        "postalCode": 10021  
    },  
    "phoneNumbers": [ "212 732-1234", "646 123-4567" ]  
}
```

```
<?  echo '{' + '"firstName":' + obj.firstName + ','  
      + '"lastName":' + obj.lastName + ','  
      ... + '}'  
?>
```


JSON Security Concerns

- JavaScript `eval()`
 - most JSON-formatted text is also **syntactically legal JavaScript code!**
 - built-in JavaScript `eval()` function **executes** code received
- **Invitation to hack:**
embed rogue JavaScript code (server-side attack),
intercept JSON data evaluation (client-side attack)
 - **Safe alternative:** `parseJSON()` method,
see ECMAScript v4 and www.json.org/json.js
- Cross-site request forgery
 - malicious page can request & obtain JSON data belonging to another site

AJAX / JSON Portability

- AJAX uses **standardized components**, supported by all major browsers:
 - JavaScript, XML, HTML, CSS
- XMLHttpRequest object part of **std DOM**
 - Windows: ActiveX control Msxml2.XMLHTTP (IE5), Microsoft.XMLHTTP (IE6)
- ...similarly for JSON

Appraisal: AJAX Advantages

- Reduced **bandwidth usage**
 - No complete reload/redraw, HTML generated locally, only actual data transferred
→ payload coming down much smaller in size
 - Can load stubs of event handlers, then functions on the fly
- **Separation** of data, format, style, and function
 - encourages programmers to clearly separate methods & formats:

Raw data / content	→ normally embedded in XML
webpage	→ HTML / XHTML
web page style elements	→ CSS
Functionality	→ JavaScript + XMLHttpRequest + server code

Appraisal: AJAX Disadvantages

- **Browser integration**
 - dynamically created page not registered in browser history
 - bookmarks
- **Search engine optimization**
 - Indexing of Ajax page contents?
 - (not specific to Ajax, same issue with all dynamic data sites)
- **Web analytics**
 - Tracking of accessing page vs portion of page vs click?
- **Response time concerns from network latency**
 - Web transfer hidden → effects from delays sometimes difficult to understand for users
- **Reliance on JavaScript**
 - JavaScript compatibility issue → blows up code;
Remedy: libraries such as **prototype**
 - IDE support used to be poor, changing
 - Can switch off JavaScript in my browser
- **Security**
 - Can fiddle with data getting into browser

Summary

- AJAX allows to add desktop flavour to web apps
 - JSON as lightweight, fast alternative to XML
- Web programming paradigm based on existing, available standards
- Issues: browser compatibility, security, web dynamics
- Many usages:
 - real-time form data validation; autocomplete; bg load on demand; sophisticated user interface controls and effects (**trees**, menus, data tables, rich text editors, calendars, progress bars, ...); partial submit; mashups (app mixing); desktop-like web app



Resources

■ Books:

- Michael Mahemoff: Ajax Design Patterns. O'Reilly, 2006
- Mark Pruet: Ajax and Web Services. O'Reilly, 2006

■ Web:

- www.openajaxalliance.org/
- w3schools.org/ajax
- Mozilla Developer Center: AJAX:Getting Started
 - developer.mozilla.org/en/docs/AJAX:Getting_Started
- www.json.org

Tool Support: Examples

- jQuery, <http://jquery.com/>

```
$( "button.continue" ).html( "Next Step..." )
```

- AJAX:

```
$.ajax({  
  url: "/api/getWeather",  
  data: {  
    zipcode: 97201  
  },  
  success: function( data ) {  
    $( "#weather-temp" ).html( "<b>" + data + "</b> degrees" );  
  }  
});
```