university of koblenz
Computer Science

4. Communication (Part II)

Engineering Web and Data-intensive Systems

Dr. Volker Riediger - Winter Term 2020/21

ist Institute for Software Technology

Communication (Part II)

- Web Services
- SOAP and REST
- Data Representation at Layer Boundaries
- Web Service Examples

Image: colourbox.de

Chapter 4.
Communication (continued)

First part of this section…

4.1 Network Communication

4.2 HTTP Protocol

4.3 Sessions

4.4 Scaling and Load Balancing

4.5 Web Services

Web Services

Web Services offer functionality (services) to clients by use of connectivity and protocols of the Internet.

- SOAP or „Big“ Web Services
  - mainly used for communication between applications on the business layer
  - technical call via HTTP(S) and SOAP protocol
  - other transport protocols possible, e.g. E-Mail instead of HTTP
  - data represented as XML documents
- RESTful Web Services
  - light-weight services, e.g. for interactive web pages
  - technical call via HTTP(S) without specific protocol
  - commonly used as target for JavaScript’s AJAX calls
  - data represented as JSON documents

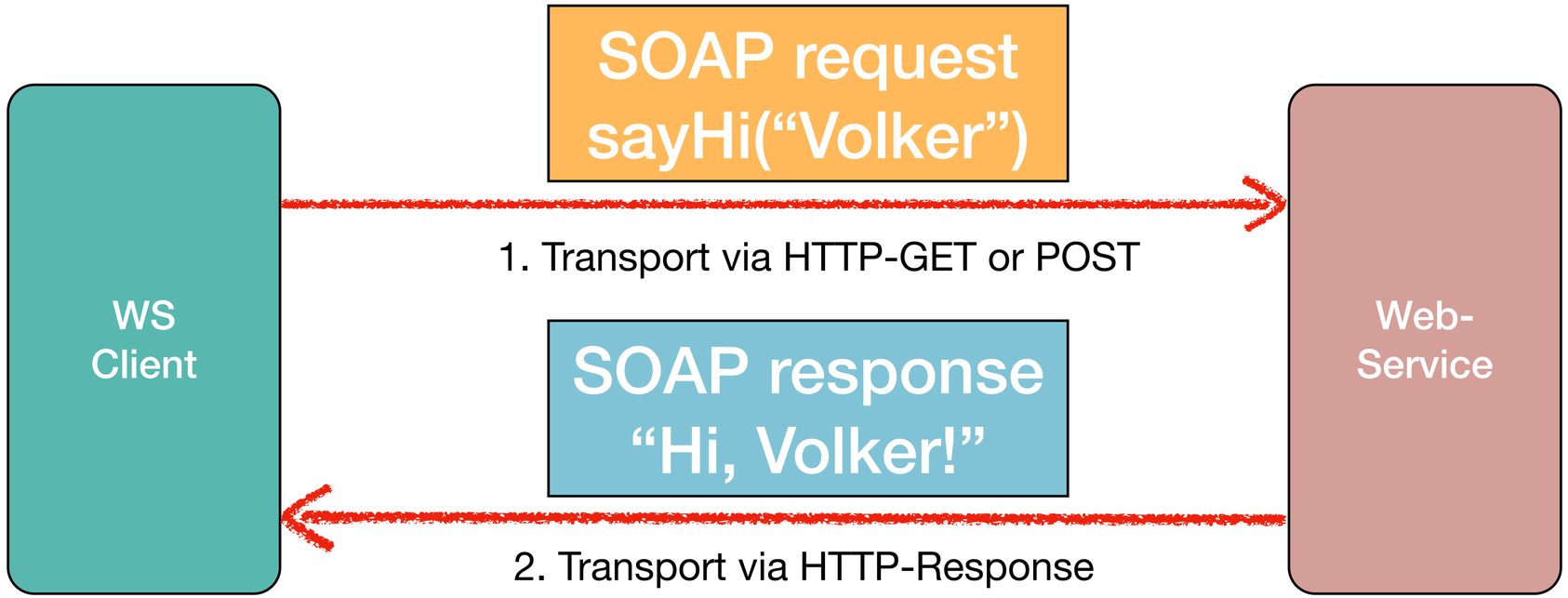
SOAP Services
“big web services”

SOAP

- Formerly called Simple Object Access Protocol;
  today SOAP is used as is, no longer as acronym
- W3C standard
- XML language for message exchange
- used for remote method calls in Web Services
- supports synchronous and asynchronous communication
- Message content specifications
  - data type definition via XML schema language
  - simple types, records, arrays
  - attachments according to MIME standard

Call of a Web-Service

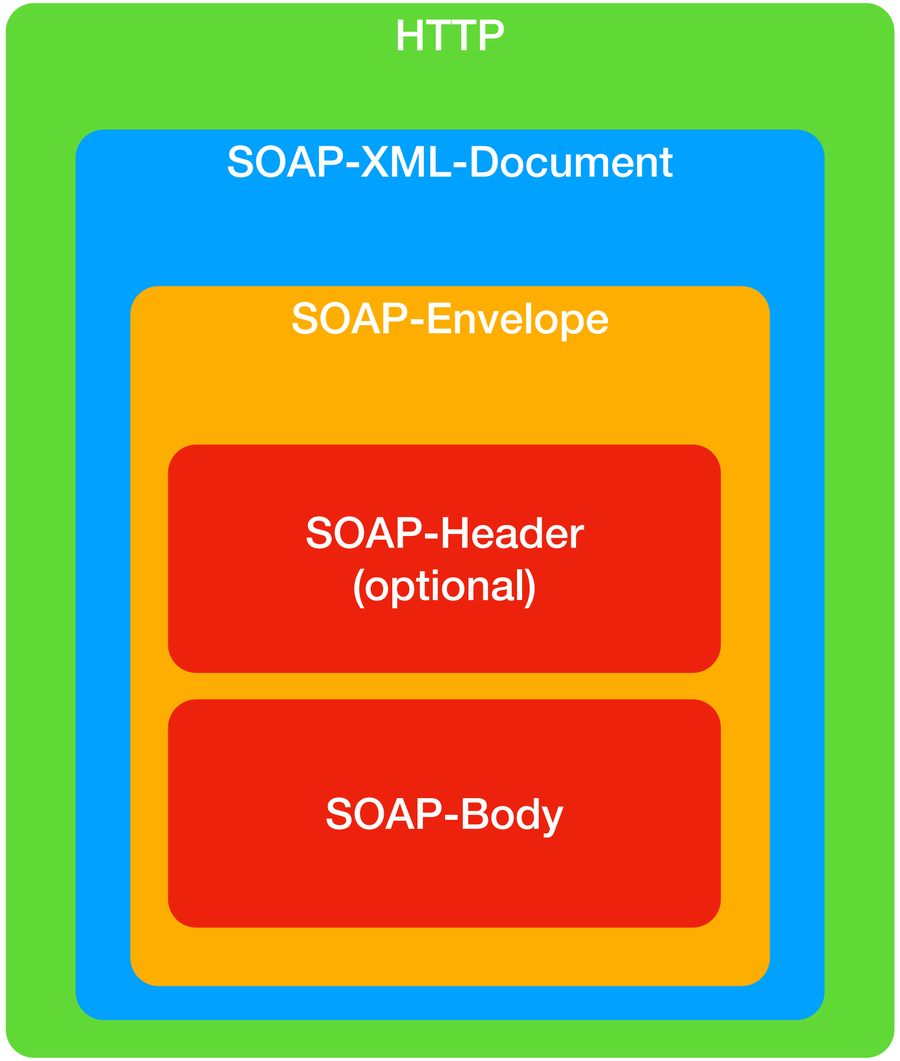
Also known as “XML-RPC” (Remote Procedure Call)



In case of errors: SOAP fault instead of SOAP response

Structure of a SOAP message

- Transport protocol
  HTTP (or others)
- Message (XML document)
  - mandatory envelope
    and body
  - optional header information for processing on final server or intermediate servers, e.g. WS security, encryption, transactions, ...
  - data payload contained in body



SOAP request

```
POST /StockQuote HTTP/1.1
Host: www.stockquoteserver.com
Content-Type: text/xml; charset="utf-8"
Content-Length: nnnn
SOAPAction: "Some-URI"

<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>

  <SOAP-ENV:Body>
    <m:GetLastTradePriceDetailed xmlns:m="Some-URI">
      <Symbol>DEF</Symbol>
      <Company>DEF Corp</Company>
      <Price>34.1</Price>
    </m:GetLastTradePriceDetailed>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

SOAP response

```
HTTP/1.1 200 OK
Content-Type: text/xml; charset="utf-8"
Content-Length: nnnn

<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>

  <SOAP-ENV:Body>
    <m:GetLastTradePriceResponse xmlns:m="Some-URI">
      <Price>34.5</Price>
    </m:GetLastTradePriceResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

SOAP response with header

```
HTTP/1.1 200 OK
 Content-Type: text/xml; charset="utf-8"
 Content-Length: nnnn

 <SOAP-ENV:Envelope
   xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
   SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
    <SOAP-ENV:Header>
        <t:Transaction
          xmlns:t="some-URI"
          xsi:type="xsd:int" mustUnderstand="1">
            5
        </t:Transaction>
    </SOAP-ENV:Header>
    <SOAP-ENV:Body>
        <m:GetLastTradePriceResponse
          xmlns:m="Some-URI">
            <Price>34.5</Price>
        </m:GetLastTradePriceResponse>
    </SOAP-ENV:Body>
 </SOAP-ENV:Envelope>
```

SOAP fault

```
HTTP/1.1 500 Internal Server Error
Content-Type: text/xml; charset="utf-8"
Content-Length: nnnn

<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
   <SOAP-ENV:Body>
       <SOAP-ENV:Fault>
           <faultcode>SOAP-ENV:Server</faultcode>
           <faultstring>Server Error</faultstring>
           <detail>
               <e:myfaultdetails xmlns:e="Some-URI">
                 <message>
                   My application didn't work
                 </message>
                 <errorcode>
                   1001
                 </errorcode>
               </e:myfaultdetails>
           </detail>
       </SOAP-ENV:Fault>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

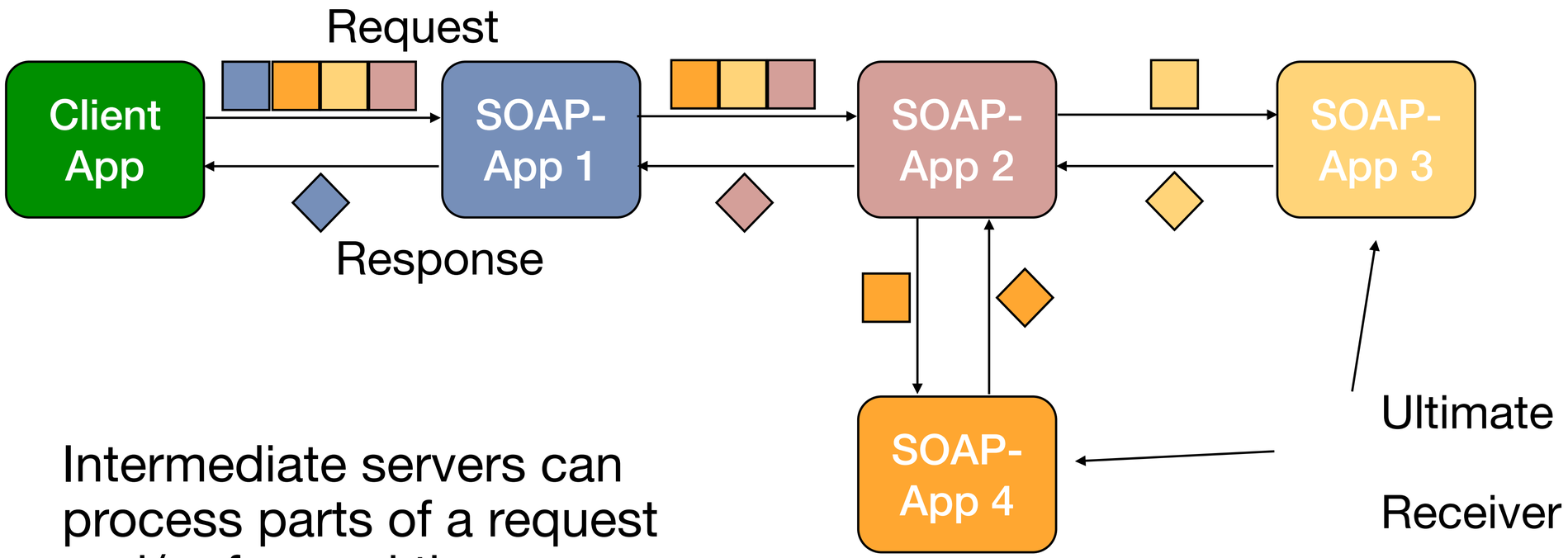
SOAP Processing

A. Identification of relevant message parts

B. Verification, ensures that all parts can be processed
   then execution or denial

C. In case of intermediate server: remove all parts of step A and forward
   message to next server

Complex SOAP processing



Intermediate servers can process parts of a request and/or forward the message to other destinations

WSDL

- **W**eb **S**ervice **D**escription **L**anguage
- XML language
- Description of web service data types and operations
- Consists of **definitions** for…
  - Types: parameter types (XML schema)
  - Message: SOAP message formats
  - Port: interface with method signatures (operations)
  - Binding: specification of protocol and encoding
  - Service: definition of the service URI

WSDL Example

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="HiWS" targetNamespace="urn:HiWS/wsdl" xmlns:tns="urn:HiWS/wsdl" xmlns="http://schemas.xmlsoap.org/wsdl/"
             xmlns:ns2="urn:HiWS/types" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/">
  <types>
    <schema targetNamespace="urn:HiWS/types" xmlns:tns="urn:HiWS/types" xmlns:soap11-enc="http://schemas.xmlsoap.org/soap/encoding/" xmlns:xsi="http://www.w3.org/200
XMLSchema-instance" xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/" xmlns="http://www.w3.org/2001/XMLSchema">
      <complexType name="sayHi">
        <sequence>
          <element name="String_1" type="string" nillable="true"/></sequence></complexType>
      <complexType name="sayHiResponse">
        <sequence>
          <element name="result" type="string" nillable="true"/></sequence></complexType>
      <element name="sayHi" type="tns:sayHi"/>
      <element name="sayHiResponse" type="tns:sayHiResponse"/></schema></types>
  <message name="HiWSSEI_sayHi">
    <part name="parameters" element="ns2:sayHi"/></message>
  <message name="HiWSSEI_sayHiResponse">
    <part name="result" element="ns2:sayHiResponse"/></message>
  <portType name="HiWSSEI">
    <operation name="sayHi">
      <input message="tns:HiWSSEI_sayHi"/>
      <output message="tns:HiWSSEI_sayHiResponse"/></operation></portType>
  <binding name="HiWSSEIBinding" type="tns:HiWSSEI">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document"/>
    <operation name="sayHi">
      <soap:operation soapAction=""/>
      <input>
        <soap:body use="literal"/></input>
      <output>
        <soap:body use="literal"/></output></operation></binding>
  <service name="HiWS">
    <port name="HiWSSEIPort" binding="tns:HiWSSEIBinding">
      <soap:address location="REPLACE_WITH_ACTUAL_URL"/></port></service></definitions>
```

Information Schema

- Description of payload data types
  - Syntax XML Schema
  - Semantics (not specified in XML, but in external documentation)
- Application specific
- Interoperability by standardization
  - e.g., RosettaNet: Supply Chain B2B models and processes
  - ebXML (Electronic Business using XML), UDDI, ...

REST Services

RESTful Web Services

- REpresentational State Transfer
- Architectural style to define web services
- Main concept of REST is that a URI represents exactly one resource.
- Different resources have different URI paths
- HTTP methods are used to execute CRUD operations on resources
  - Create by means of `POST` or `PUT`
  - Read by means of `GET`
  - Update by means of `PUT`, `POST`, or `PATCH`
  - Delete by means of `DELETE`
- Content consumed/delivered by requests may be encoded in different formats, e.g. HTML, XML, JSON, …

Activities in **form based** Web Applications

BROWSER

1. Request of a page (`GET`)

3. Request for further resources (`GET`)
   e.g. images, scripts, styles…

5. Rendering (presentation) of the complete web page

6. Send form data (`POST`)

WEB SERVER

2. Delivery of HTML document

4. Delivery of requested data

steps 3 and 4 possibly repeated many times

7. Processing of form data, then delivery of response document (continue at step 2)

Activities in **Single Page** Web Applications

BROWSER

1. Request of a page (`GET`)

3. Request for further resources (`GET`)
   e.g. images, scripts, styles…

5. Rendering (presentation) of the complete web page

6. Issue AJAX-Requests (`GET`/`POST`/`PUT`/`DELETE`)

8. Processing of response, modification of the page

WEB SERVER

2. Delivery of HTML document

4. Delivery of requested data

steps 3. and 4. possibly repeated many times

7. Processing of request, response as XML or JSON document, „pure data“

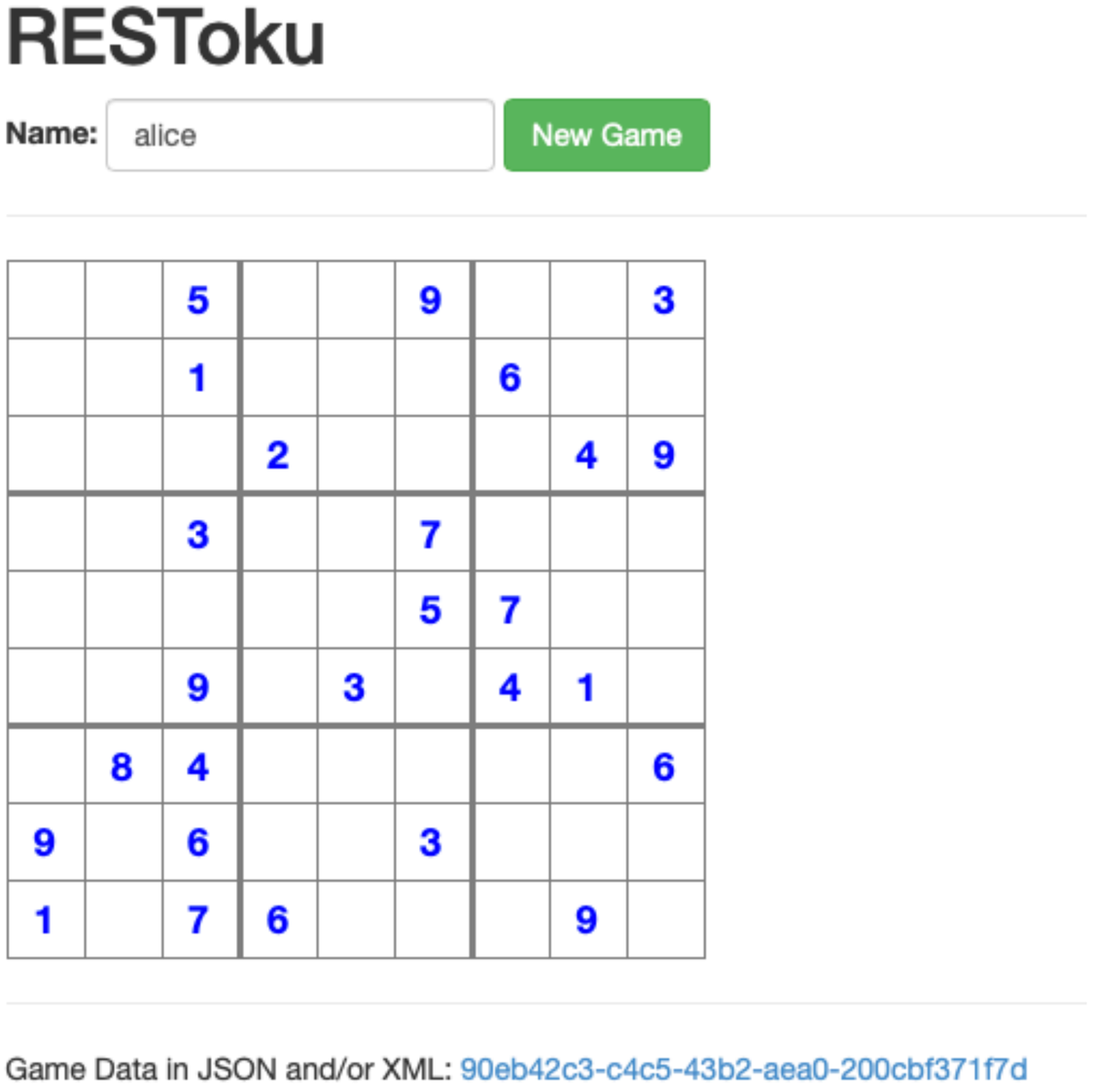
steps 6-8 possibly repeated many times

RESTful Web Services

- REST = REpresentational State Transfer
- Main concept of REST:
  URL represents exactly one resource.
- Different resources have different URL paths

- HTTP methods are used to execute CRUD operations on resources
  - Create by means of `PUT`
  - Read by means of `GET`
  - Update by means of `POST`
  - Delete by means of `DELETE`

Example Application „RESToku”



RESToku Service

HTTP request	URL	Purpose
PUT	/newgame/{name}	creates a new game
GET	/games	lists current games
GET	/games/{uuid}	state of a game
GET	/games/{uuid}/{index}	value of a field
POST	/games/{uuid}/{index}/{value}	change value of a field
DELETE	/games/{uuid}	removes a game

RESTful Web Services

- REST is not a product, nor a standard, but an architectural model
- Frameworks for REST implementations exist in many (Web-) platforms, e.g.
  - JavaEE
  - Ruby on Rails
  - PHP
  - Grails
  - Node.js
  - …

RESTful Web Services

- Properties of RESTful Web Services
  - Addressability - A RESTful Web Service has a unique address.
  - Different representations - different data representations or transport formats can be offered
  - Stateless - like the HTTP(S) protocol, REST assumes a stateless client-server-protocol
  - Operations - a RESTful Web Service must offer CRUD operations for resources
  - Use of hyperlinks - hyperlinks can be used to connect different resources

RESTful Web Services - XML

- Content presented in machine readable form
- XML often used as data exchange format

<?xml version="1.0" encoding="UTF-8" standalone="yes"?><gameState><errorCount>0</errorCount><fieldCount>26</fieldCount><game>0</
game><game>0</game><game>5</game><game>0</game><game>0</game><game>9</game><game>0</game><game>0</game><game>3</game><game>0</
game><game>0</game><game>1</game><game>0</game><game>0</game><game>0</game><game>6</game><game>0</game><game>0</game><game>0</
game><game>0</game><game>0</game><game>2</game><game>0</game><game>0</game><game>0</game><game>4</game><game>9</game><game>0</
game><game>0</game><game>3</game><game>0</game><game>0</game><game>7</game><game>0</game><game>0</game><game>0</game><game>0</
game><game>0</game><game>0</game><game>0</game><game>0</game><game>5</game><game>7</game><game>0</game><game>0</game><game>0</
game><game>0</game><game>9</game><game>0</game><game>3</game><game>0</game><game>4</game><game>1</game><game>0</game><game>0</
game><game>8</game><game>4</game><game>0</game><game>0</game><game>0</game><game>0</game><game>0</game><game>6</game><game>9</
game><game>0</game><game>6</game><game>0</game><game>0</game><game>3</game><game>0</game><game>0</game><game>0</game><game>1</
game><game>0</game><game>7</game><game>6</game><game>0</game><game>0</game><game>0</game><game>9</game><game>0</game><name>alice</
name><state>0</state><state>0</state><state>1</state><state>0</state><state>0</state><state>1</state><state>0</state><state>0</
state><state>1</state><state>0</state><state>0</state><state>1</state><state>0</state><state>0</state><state>0</state><state>1</
state><state>0</state><state>0</state><state>0</state><state>0</state><state>0</state><state>1</state><state>0</state><state>0</
state><state>0</state><state>1</state><state>1</state><state>0</state><state>0</state><state>1</state><state>0</state><state>0</
state><state>1</state><state>0</state><state>0</state><state>0</state><state>0</state><state>0</state><state>0</state><state>0</
state><state>0</state><state>1</state><state>1</state><state>0</state><state>0</state><state>0</state><state>0</state><state>1</
state><state>0</state><state>1</state><state>0</state><state>1</state><state>1</state><state>0</state><state>0</state><state>1</
state><state>1</state><state>0</state><state>0</state><state>0</state><state>0</state><state>0</state><state>1</state><state>1</
state><state>0</state><state>1</state><state>0</state><state>0</state><state>1</state><state>0</state><state>0</state><state>0</
state><state>1</state><state>0</state><state>1</state><state>1</state><state>0</state><state>0</state><state>0</state><state>1</
state><state>0</state><uuid>90eb42c3-c4c5-43b2-aea0-200cbf371f7d</uuid></gameState>

RESTful Web Services - JSON

- JavaScript Object Notation
- Any valid JSON document is at the same time a valid JavaScript fragment that can be interpreted by a function `JSON.parse(…)` or `eval()`
- In contrast to XML: reduced overhead, less memory consumption, less transfer volume, less bandwidth usage
- Simple syntax, easy to generate and to process


```
{"errorCount":0,"fieldCount":26,"game":
[0,0,5,0,0,9,0,0,3,0,0,1,0,0,0,6,0,0,0,0,0,2,0,0,0,4,9,0,0,3,0,0,7,0,0,0,0,0,0,0,0,5,7,0,0,0,0,9,0,3,0,4,1,0,0,8,4,0,0,0,0,0,6,9,0,6,0,0,3
,0,0,0,1,0,7,6,0,0,0,9,0],"name":"alice","state":
[0,0,1,0,0,1,0,0,1,0,0,1,0,0,0,1,0,0,0,0,0,1,0,0,0,1,1,0,0,1,0,0,1,0,0,0,0,0,0,0,0,1,1,0,0,0,0,1,0,1,1,0,0,1,1,0,0,0,0,0,1,1,0,1,0,0,1
,0,0,0,1,0,1,1,0,0,0,1,0],"uuid":"90eb42c3-c4c5-43b2-aea0-200cbf371f7d"}
```

RESTful Web Services - JSON

- JSON data types:
  - Undefined value represented as `null`
  - Strings in double quotes
  - Signed numbers as sequence of digits `0-9`
  - Boolean values `true` and `false`
  - Arrays as comma separated lists enclosed in square brackets `[ ]`
  - Objects start `{` and end with `}` and consist of comma-separated key-value-pairs
    - key is a String
    - value can have an arbitrary type

XMLHttpRequest

- RESTful Web Services are used via JavaScript `XMLHttpRequest`
- Advantage: Data can be loaded dynamically without need to reload the complete page.
- XMLHttpRequests in general use asynchronous communication - the script doesn't have to wait for the response

- All HTTP methods can be used
- Current specification: XMLHttpRequest Level 2
  - timeouts
  - transfer of binary data
  - security features
- Web browsers support XMLHttpRequest via embedded JavaScript interpreters (XHR objects).

WADL

Web Application Description Language

- Resembles WSDL, but much simpler format
- A standard specification was submitted to W3C: https://www.w3.org/Submission/wadl/
- WADL is only one of a variety of REST API specification languages (see for example https://en.wikipedia.org/wiki/Overview_of_RESTful_API_Description_Languages)
- Java REST endpoint implementations using the JAX-RS specification (“Jersey”) automatically create WADL descriptions
- E.g., the RESToku service shown in part 1 of this lesson can be queried for it’s WADL
- https://localhost:8181/RESToku/resources/application.wadl

  (this link will not work on your machine…)

Example WADL response

```
▼<application xmlns="http://wadl.dev.java.net/2009/02">
   <doc xmlns:jersey="http://jersey.java.net/" jersey:generatedBy="Jersey: 2.30.payara-p1 2020-01-23 15:17:46"/>
   <doc xmlns:jersey="http://jersey.java.net/" jersey:hint="This is simplified WADL with user and core resources only. To get full WADL with
   parameter detail. Link: https://localhost:8181/RESToku/resources/application.wadl?detail=true"/>
  ▼<grammars>
   ▼<include href="application.wadl/xsd0.xsd">
      <doc title="Generated" xml:lang="en"/>
    </include>
   </grammars>
  ▼<resources base="https://localhost:8181/RESToku/resources/">
   ▼<resource path="/">
    ▼<resource path="/newgame/{name}">
       <param xmlns:xs="http://www.w3.org/2001/XMLSchema" name="name" style="template" type="xs:string"/>
      ▼<method id="getNewGameAsJson" name="PUT">
       ▼<response>
          <ns2:representation xmlns:ns2="http://wadl.dev.java.net/2009/02" xmlns="" element="gameState" mediaType="application/json"/>
          <ns2:representation xmlns:ns2="http://wadl.dev.java.net/2009/02" xmlns="" element="gameState" mediaType="application/xml"/>
          <ns2:representation xmlns:ns2="http://wadl.dev.java.net/2009/02" xmlns="" element="gameState" mediaType="text/xml"/>
        </response>
       </method>
     </resource>
    ▼<resource path="/games/{uuid}/{index}">
       <param xmlns:xs="http://www.w3.org/2001/XMLSchema" name="index" style="template" type="xs:int"/>
       <param xmlns:xs="http://www.w3.org/2001/XMLSchema" name="uuid" style="template" type="xs:string"/>
      ▼<method id="getValue" name="GET">
       ▼<response>
          <representation mediaType="text/plain"/>
        </response>
       </method>
     </resource>
    ▼<resource path="/games/{uuid}">
       <param xmlns:xs="http://www.w3.org/2001/XMLSchema" name="uuid" style="template" type="xs:string"/>
       <method id="deleteGame" name="DELETE"/>
      ▼<method id="getGameAsJson" name="GET">
       ▼<response>
          <ns2:representation xmlns:ns2="http://wadl.dev.java.net/2009/02" xmlns="" element="gameState" mediaType="application/json"/>
          <ns2:representation xmlns:ns2="http://wadl.dev.java.net/2009/02" xmlns="" element="gameState" mediaType="application/xml"/>
          <ns2:representation xmlns:ns2="http://wadl.dev.java.net/2009/02" xmlns="" element="gameState" mediaType="text/xml"/>
        </response>
       </method>
     </resource>
```

4.6 Data Representation

Data Representation

- In Web Applications, data items of different types, structure, size, and lifetime have to be stored and transmitted.
- Depending on the data properties, an appropriate data format has to be chosen, e.g.
  - Hypertext — HTML, XHTML
  - Formatting directives — CSS
  - Parameters of web services — XML, JSON
  - Images — JPG, PNG, …
  - Vector Graphics — SVG
  - Formatted (read-only) documents — PDF
  - …and many more

MIME and Media Types

- Web applications (such as web servers) use so-called Media Types (formerly called MIME Types) to announce and/or negotiate the accepted formats
- MIME = Multipurpose Internet Mail Extensions
  Standardized list of data formats that help sending, receiving, and handling of non-textual as well as textual content
- Media Type registry run by IANA (Internet Assigned Numbers Authority)

Serialization

- In the Web, all documents are character streams.
  → Need for general approach for serializing data.

- Serialization transforms objects and/or data structures into a sequential (byte stream) format.

- Can be stored / exchanged and reconstructed elsewhere.
  → A serialization can be used to create a semantically equivalent clone somewhere else.

- Standard way for serialization in the Web is XML (eXtensible Markup Language)

- Current version: 1.1, but 1.0 still most widely used.

Short Time Serialization - Marshalling

- Serialization occurs when communicating parties, e.g. web services, exchange information.
  - Using a web service, a server side function is executed.
  - Such functions take parameter values and may have execution result values
  - There’s a need to transfer the parameters and the results between client and server
  - To achieve this, the values are serialized by the sender and de-serialized by the receiver
  - De-serialization means converting the character stream back to internal object representations
  - Parameter passing by serialization is called Marshalling

Marshalling Example

1. parameter serialization, e.g. to XML

```
<?xml version="1.1"?>
<convertParam>

  <amount>42.00</amount>

  <sourceCurrency>EUR</sourceCurrency>

  <targetCurrency>USD</targetCurrency>

</convertParam>
```

2. de-serialization, to internal objects

Client

42.00€

46.20$

Currency Conversion Service

convert(amount, source, target) → amount, currency

3. perform function

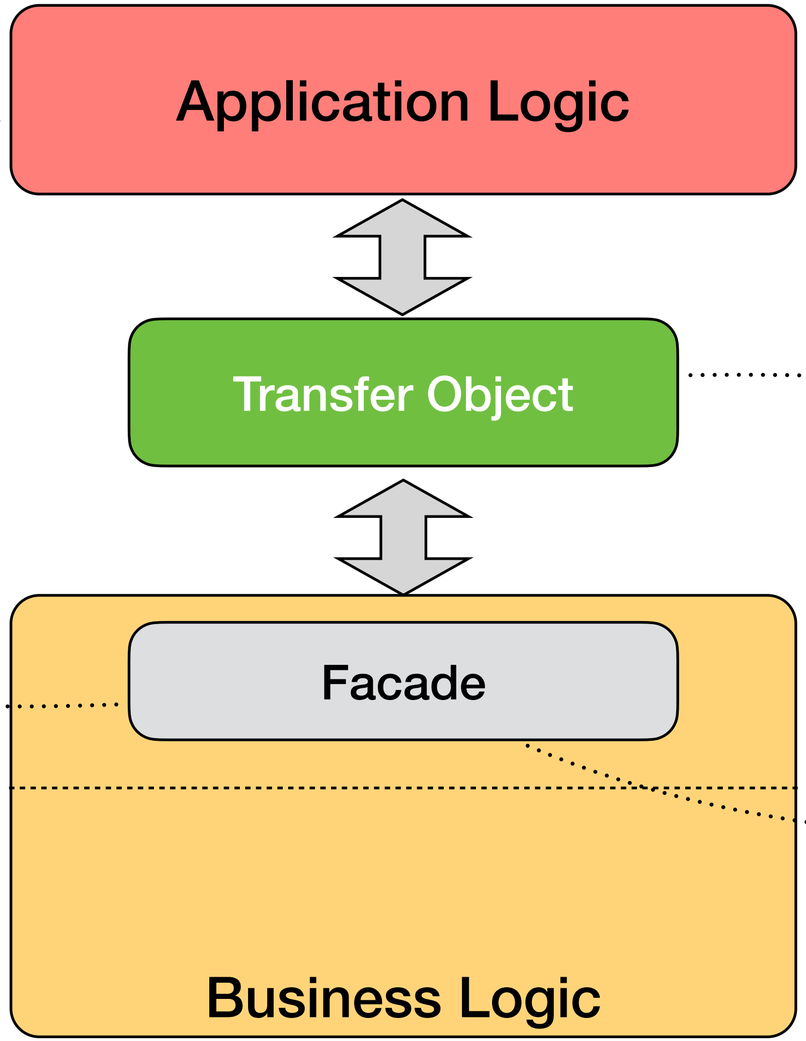
```
<?xml version="1.1"?>

<convertResult>

  <amount>46.20</amount>

  <currency>USD</currency>

</convertResult>
```

4. result serialization

5. de-serialization, to internal objects

Relation of XML/JSON to Application Architecture

Uses BL functionality to realize application functions



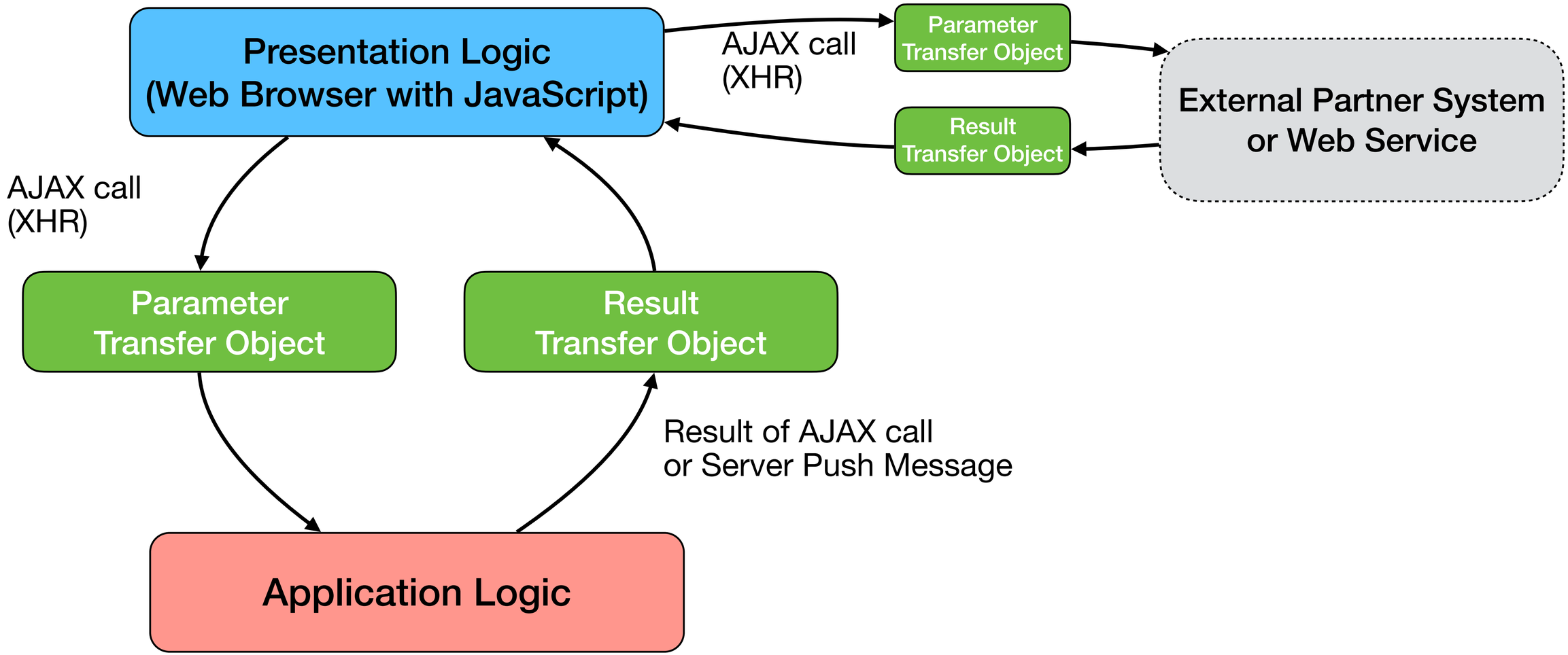
Different DTOs allow for arbitrary level of detail.

Facade converts between BL-internal and external data formats, e.g. binary, **XML, JSON**…

Service interface relies on operation-specific transfer objects.

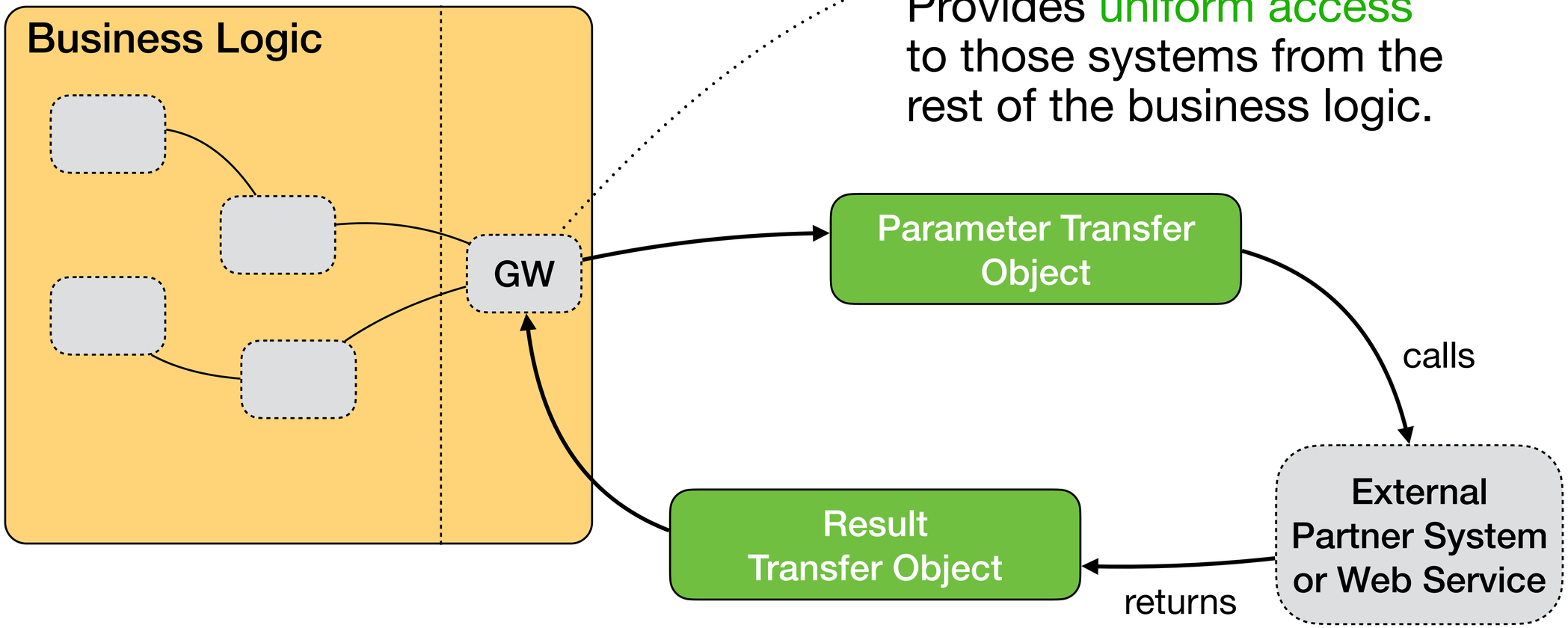
Services exposed as SOAP, REST or other RPC technology

Transfer Objects between Browser/PL and AL and/or Web Service



Transfer Objects at Gateways

Encapsulates functionality of external systems. Provides uniform access to those systems from the rest of the business logic.



XML
eXtensible Markup Language

XML: General Idea

- Uses ‚generalized markups‘ which are
  - declarative (describing structure and text attributes)
  - rigorous (being processable by automated tools).
- Does not come with pre-defined markups or implicit semantics (in contrast to HTML).
- Allows the definition of application-specific markups.

XML: Interoperability

- XML descriptions are
  - serialized (transferable as byte streams)
  - platform independent
  - tractable by generic tools
  - readable by humans
- … thus, supporting interoperability of different technologies.

- Note: Serialization is the lowest level of interoperability!

XML Constituents

- XML-text is (usually Unicode-)text consisting of
  - markups and
  - contents.

```
<?xml version="1.0"?>
<order OrderID="10643">
    <item><book isbn="123-321" /></item>
    <item><cdrom title="Vivaldi Four Seasons" /></item>
    <item><book isbn="3-8265-8059-1" /></item>
    <OrderDate ts="2003-06-30T00:00:00" />
    <price>167.00 EUR</price>
</order>
```

XML Elements, Attributes and Entities

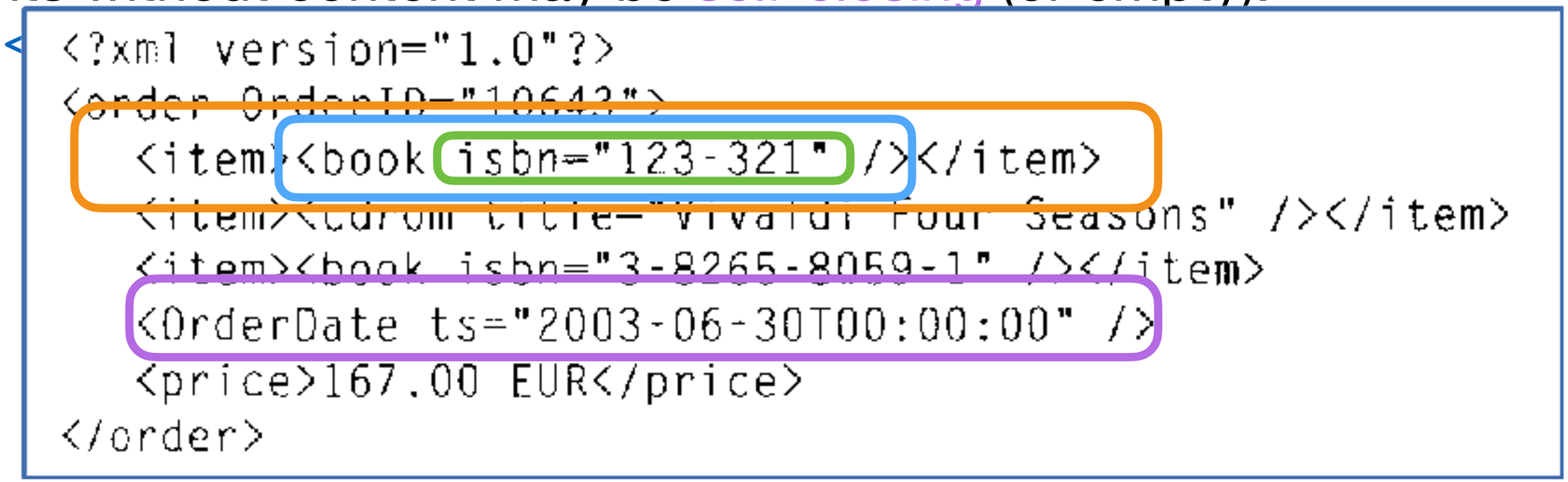
- XML contains:
  - Elements: The main building blocks.
  - Attributes: Provide extra information about elements. Placed inside the starting tag of an element.
  - Entit

```
<?xml version="1.0"?>
<order OrderID="10643">
   <item><book isbn="123-321" /></item>
   <item><cdrom title="Vivaldi Four Seasons" /></item>
   <item><book isbn="3-8265-8059-1" /></item>
   <OrderDate ts="2003-06-30T00:00:00" />
   <price>167.00 EUR</price>
</order>
```

&...;

Elements and Attributes

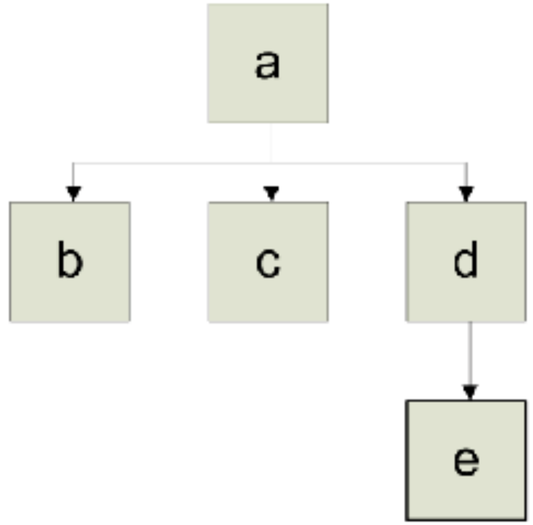
- Elements: text parts marked by opening and closing tags:
  `<tag> … </tag>` and can be nested
- Elements may have attributes in their starting tag:
  `<tag attr1=value1 … attrk=valuek > … </tag>`
- Elements without content may be self-closing (or empty):



Elements: Tree Structure

- Elements may be nested.
- Thus, they have a tree-like structure
  thereby, one can speak of parent, child, sibling, root, and leaf nodes

```
<a>
    <b> ... </b>
    <c> ... </c>
    <d>
        <e> ... </e>
    </d>
</a>
```



Attributes

- Attributes are name-value pairs.
- Each name may appear only once in a tag, i.e. have mapping
  map: name → value
- All values are atomic (strings).
- If lists of values are needed, can provide e.g. as comma-separated-values (csv), but transparent for XML.

Entities

- Forbidden characters are encoded as entities.
- Only five entities predefined, which are referenced as:

`<`	<
`>`	>
`&`	&
`&apos;`	‘
`&quot;`	“

- Further entities may be defined in DTDs.

- Unicode-characters also referenced as entities:
  - `&#nnnn;` (where n is a hexadecimal number)
- Example: `&#103B;` stands for greek ‘α’

Documents

- XML-documents start with a declaration
  `<?xml version="1.0" encoding="utf-8"?>`
  („encoding“ is optional)

- and contain one single outer root element (here: <order>):

```
<?xml version="1.0"?>
<order OrderID="10643">
   <item><book isbn="123-321" /></item>
   <item><cdrom title="Vivaldi Four Seasons" /></item>
   <item><book isbn="3-8265-8059-1" /></item>
   <OrderDate ts="2003-06-30T00:00:00" />
   <price>167.00 EUR</price>
</order>
```

Well-formed Documents

- Rules for well-formed XML documents
  - XML tags are case sensitive
  - All XML elements must be balanced (have closing tag)
  - XML elements must be properly nested
  - XML documents must have one root element
  - XML attribute values must be quoted
  - Comments are enclosed in `<!-- ... -->`
  - There are five entity references:
    `&lt; &gt; &amp; &apos; &quot;`
  - Those entity references have to be used whenever the respective characters are part of non-markup, i.e. text content, comments, or attribute values.

XML Dialects

- Applications need semantics

- XML dialects may be specified by
  - a DTD or
  - an XML Schema file
  - (cf later for details).

- Examples of XML dialects include:
  - XMI, (X)HTML, RSS, Atom

What’s a name space and what is it used for?
Do you know examples?

Name Space Examples

- The concept of name spaces is widely used in Computer Science, e.g.:
  - C++: x::a Picture::draw()
  - file paths: x/a Users/john/pictures
  - domain names: a.x info.company.net
  - Java: x.a Person.name
  - and in mathematics: x_a

XML Name Spaces

- To allow merging of documents from different sources, name spaces are introduced (to solve conflicts between tags).

- A name space is a named abstract container for items with different names (i.e., a set).

- Names spaces are used to avoid collisions of equally named elements, since they can be disambiguated by tagging them with their name space identifier.

XML Name Spaces

- A name space is a set of identifiers
  ns : set of name

- Example:
  - Given two namespaces `x = {a,b,c}` and `y = {a,c,d,e}`,
  - the occurrences of the names `a,…,e` can be disambiguated by tagging (qualifying) them with their name space,
    e.g., `x:a` vs. `y:a`.

Example

```
<table>
  <tr>
    <td>Apples</td>
    <td>Bananas</td>
  </tr>
</table>
```

This XML carries information about a table (a pi

```
<table>
  <name>African Coffee Table</name>
  <width>80</width>
  <length>120</length>
</table>
```

```
<root>

<h:table xmlns:h="http://www.w3.org/TR/html4/">
  <h:tr>
    <h:td>Apples</h:td>
    <h:td>Bananas</h:td>
  </h:tr>
</h:table>

<f:table xmlns:f="http://www.w3schools.com/furniture">
  <f:name>African Coffee Table</f:name>
  <f:width>80</f:width>
  <f:length>120</f:length>
</f:table>

</root>
```

Name Spaces

- In XML name spaces are used to disambiguate tags in different vocabularies.
- A name space may have a name and denotes an identifier (e.g., a URI) which is expected to be unique.
- Name spaces are added to XML elements by the reserved attribute `xmlns` in two forms
  - `xmlns` applies to the whole element (incl. children) and denotes the „default namespace“
  - `xmlns:prefix` applies only to those (sub)elements, which are explicitly tagged with prefix.

Example: Name Spaces

`xmlns`

```
1 <order xmlns=”uri:order”>
2   <item>
3     <book isbn=”123456”/>
4   <item/>
5 <order/>
```

`xmlns:prefix`

```
1 <o:order xmlns:o="uri:order”>
2   <o:item>
3     <o:book isbn=”123456”/>
4   <o:item/>
5 <o:order/>
```

Example: Name Spaces

`xmlns:o`

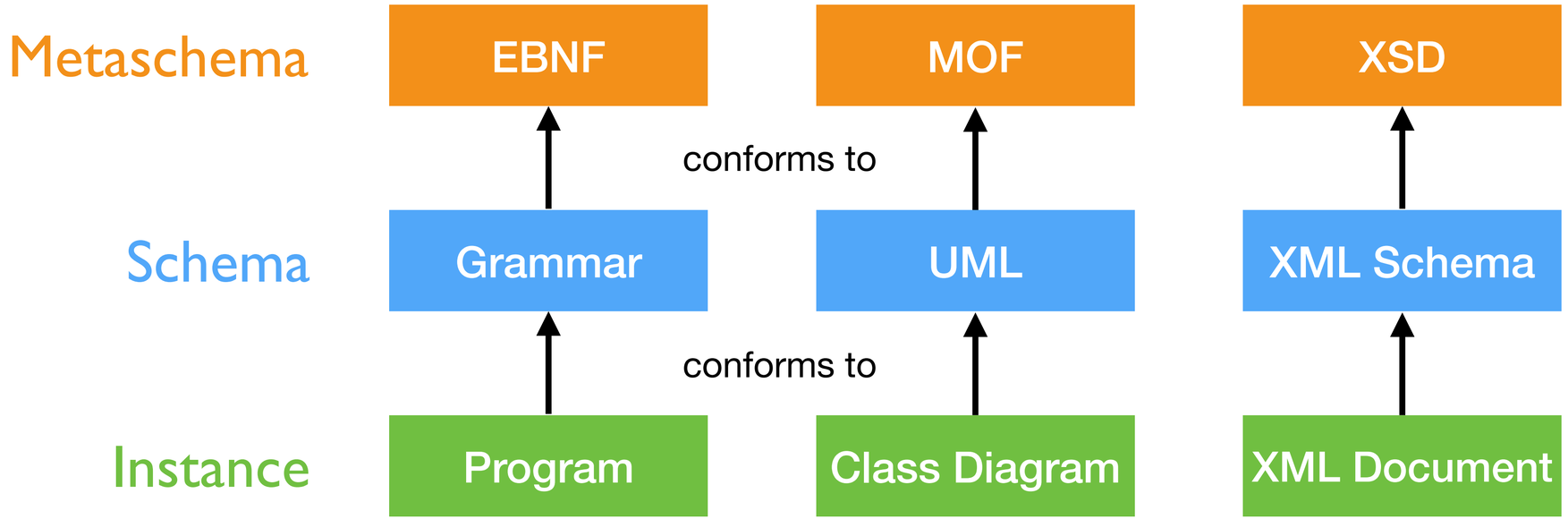
is not applied to all elements!

```
<?xml version="1.0" encoding="UTF-8"?>
<o:order  xmlns:o="http://www.example.com/order">
    <o:id>2343</o:id>
    <o:currency>JPY</o:currency>
    <o:quantity>100</o:quantity>
    <o:date>2016-05-07</o:date>
    <brokerInfo  xmlns="http://www.example.com/brokerInfo">
        <id>13456</id>
    </brokerInfo>
</o:order>
```

XML Schemas

I-S-M Dimensions

- XML Schemas help to define XML languages, much like grammars define programming languages, and meta models define modeling languages.
- Meta schemas, in turn, define schema languages
- XML Schema Definition Language (XSD) represents the XML meta schema



DTD vs. XSD

- Predecessor of XSD was the Document Type Definition (DTD)
- DTDs define markup declarations
  - elements and attributes
  - document structure (nesting)
  - basic types (text, id, idref)
  - entities
  - DTDs have a special, non-XML format
- XSDs are more expressive (and way more complicated)
- XSD Language enables
  - complex element data types and attribute value types
  - id scopes, name spaces
  - facilities to combine schemas
  - various constraints
- XSD schemas are XML documents

DTDs

- A Document Type Definition (DTD) is a meta-description that defines a document type, i.e., the structure of a class of XML documents.
- DTDs are based on regular expressions.
- This DTD language is inherited from SGML.
- Although it is still quite often being used, it is deprecated:
- Requires a separate notation (as opposed to XSD),which adds complexity
- HTML5 not based on SGML and thus no official DTD for it

DTD Example

```
<?xml version="1.0"?>
<order OrderID="10643">
   <item><book isbn="123-321" /></item>
   <item><cdrom title="Vivaldi Four Seasons" /></item>
   <item><book isbn="3-8265-8059-1" /></item>
   <OrderDate ts="2003-06-30T00:00:00" />
   <price>167.00 EUR</price>
</order>
```

```
<?xml version="1.0"?>
<!DOCTYPE order [
<!ELEMENT order (item+,OrderDate,price)>
<!ATTLIST order OrderID ID #REQUIRED>
<!ELEMENT item (book,cdrom)+>
<!ELEMENT book EMPTY>
<!ATTLIST book isbn CDATA #REQUIRED>
<!ELEMENT cdrom EMPTY>
<!ATTLIST cdrom title CDATA #REQUIRED>
<!ELEMENT OrderDate EMPTY>
<!ATTLIST OrderDate ts CDATA '2003-06-30T00:00:00'>
<!ELEMENT price (#PCDATA)>
]>
```

[source: Kappel2006WE]

(XSDs are only presented by example)

DOM
Document Object Model

Motivation

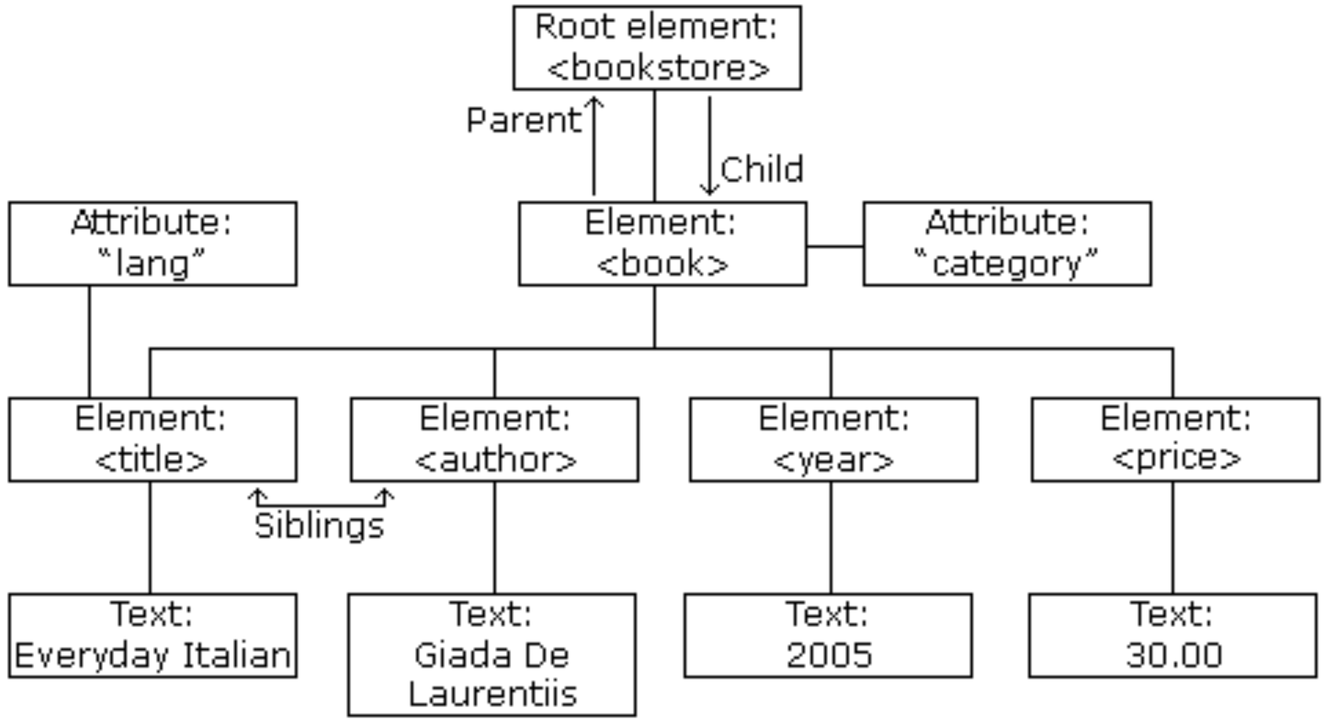
- The Document Object Model (DOM) is an application programming interface (API) for well-formed XML documents.
- It is a standard for representing XML documents as object trees (DOM trees) in object-oriented software (e.g. inside browsers).
- With DOM, programmers can build documents, traverse their structure, and add, modify, or delete elements and content.
- DOM has implementations for programming languages, e.g. the DOM API in Java

XML DOM

- DOM is a language-independent convention for representing and interacting with the constituents of XML-documents.
- DOM is a W3C recommendation (Current: Level 3, 2004) [https://www.w3.org/TR/DOM-Level-3-Core/], which specifies DOM in detail.
- The DOM tree may be created by an XML parser.

DOM Example

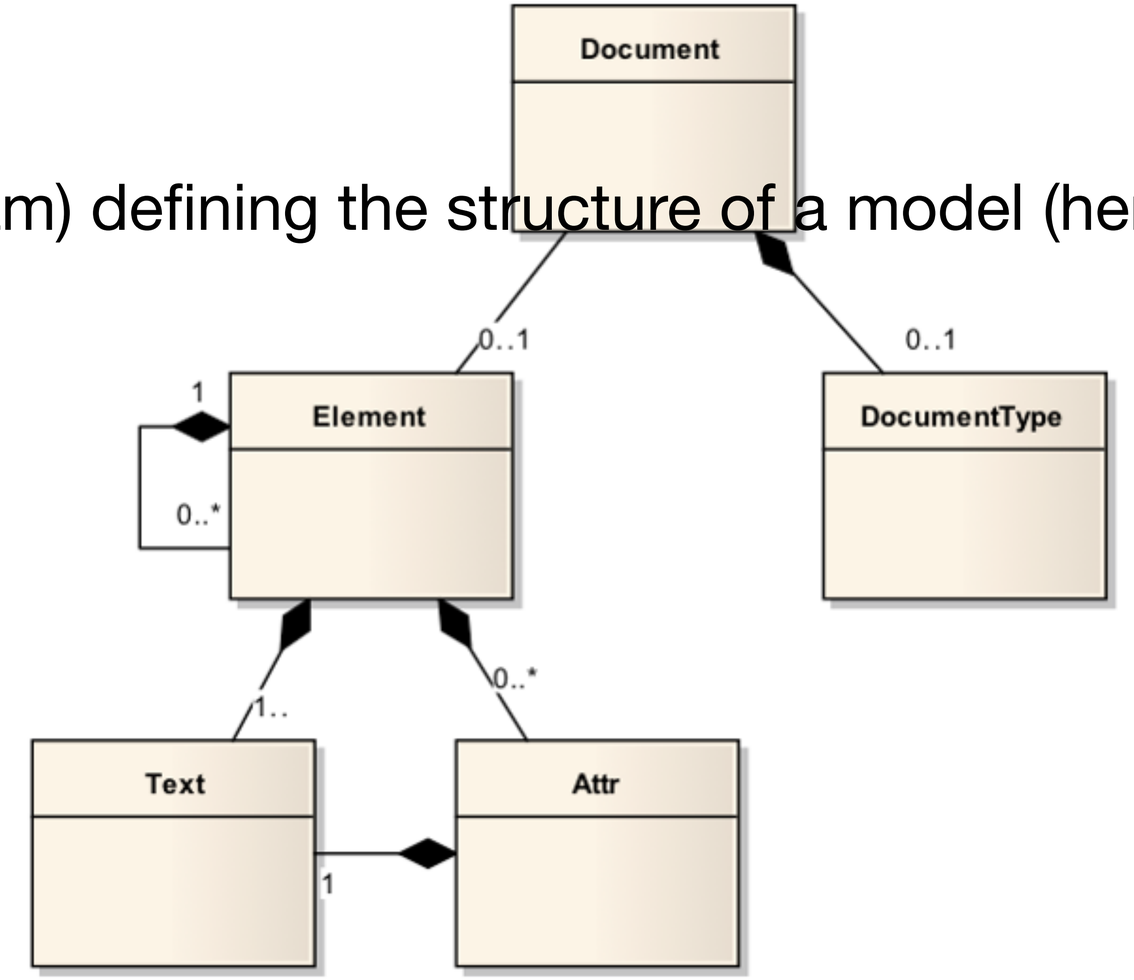
```xml
<?xml version="1.0" encoding="UTF-8"?>
<bookstore>
  <book category="cooking">
    <title lang="en">Everyday Italian</title>
    <author>Giada De Laurentiis</author>
    <year>2005</year>
    <price>30.00</price>
  </book>
  <book category="children">
    <title lang="en">Harry Potter</title>
    <author>J K. Rowling</author>
    <year>2005</year>
    <price>29.99</price>
  </book>
  <book category="web">
    <title lang="en">XQuery Kick Start</title>
    <author>James McGovern</author>
    <author>Per Bothner</author>
    <author>Kurt Cagle</author>
    <author>James Linn</author>
    <author>Vaidyanathan Nagarajan</author>
    <year>2003</year>
    <price>49.99</price>
  </book>
```



DOM Metamodel (simplified)

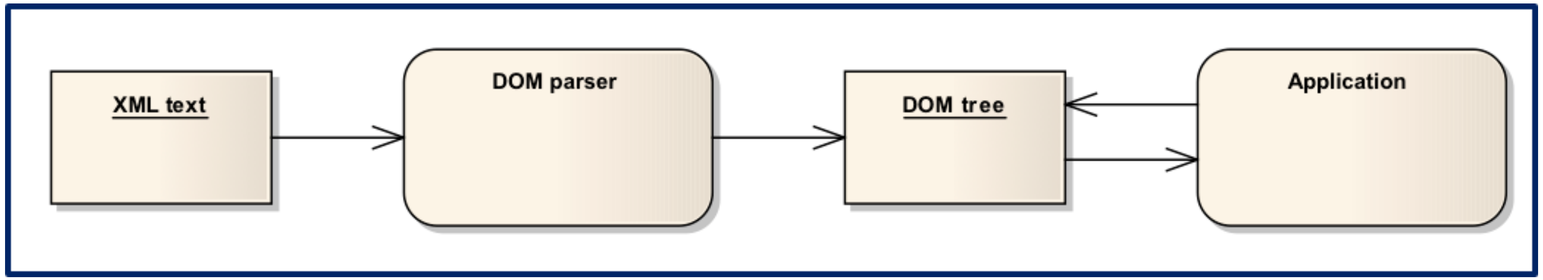
- Metamodel:

  Model (here: UML class- diagram) defining the structure of a model (here: the DOM model)



DOM Tree and JavaScript

- Web browsers parse the (X)HTML text of a page and build a DOM tree of the document



- JavaScript has built-in DOM support
- DOM modification is the base of single-page web applications
  - AJAX requests (also known as XHR - XML HTTP Request) retrieve data from servers
  - AJAX = Asynchronous Javascript And XML
  - Results are used to change or populate the DOM tree and thereby to display new or changed data
  - AJAX often uses REST APIs which no longer use XML but rather exchange JSON representations

JSON
JavaScript Object Notation

JSON

- JSON, the JavaScript Object Notation, is a textual format for data exchange
- The notation resembles the syntax for objects in the JavaScript (aka ECMAScript) programming language
- The format is standardized in RFC 8259
- Syntax is simple and only uses very few rules
- In contrast to XML, JSON has less overhead, but is also less expressive

JSON documents

- Documents are character sequences encoded in Unicode UTF-8
- JSON contains
  - primitive types like strings and numbers
  - complex types like arrays and objects
  - objects are represented by lists of key-value pairs
  - complex types can be nested

Example JSON Object


```
{
  "uuid": "9dbcac54-db55-4d24-88b2-74a0d75a68c4",
  "number": "26900900",
  "shortname": "KOBLENZ UP",
  "longname": "KOBLENZ UP",
  "km": 1.91,
  "agency": "WSA KOBLENZ",
  "longitude": 7.457699265736656,
  "latitude": 50.278472281764,
  "water": {
    "shortname": "MOSEL",
    "longname": "MOSEL"
  },
  "timeseries": [
    {
      "shortname": "W",
      "longname": "WASSERSTAND ROHDATEN",
      "unit": "cm",
      "equidistance": 15,
      "currentMeasurement": {
        "timestamp": "2020-01-30T11:45:00+01:00",
        "value": 302.0,
        "trend": 1,
        "stateMnwMhw": "unknown",
        "stateNswHsw": "unknown"
      },
      "gaugeZero": {
        "unit": "m. ü. NHN",
        "value": 58.014,
        "validFrom": "2014-01-01"
      }
    }
  ]
}
```

JSON Schema

- Adoption of the format requires to define data formats and to validate content
  - With JSON only, this is not possible
  - Currently, JSON Schema is being developed
  - JSON Schemas are special JSON documents
- This enables the definition of JSON sublanguages, much like domain specific XML languages
- Validation of documents beyond the basic syntax rules is possible

4.7 Web Service Examples

Example: Exchange Rate Service

- https://exchangeratesapi.io
- e.g., using curl:
  `curl -X GET https://api.exchangeratesapi.io/latest`
- REST API (only GET allowed)
- JSON response


```
{
   "base" : "EUR",
   "date" : "2021-01-06",
   "rates" : {
      "AUD" : 1.5824,
      "BGN" : 1.9558,
      "BRL" : 6.5119,
      "CAD" : 1.564,
      "CHF" : 1.0821,
      "CNY" : 7.9653,
      "CZK" : 26.145,
      "DKK" : 7.4393,
      "GBP" : 0.90635,
      "HKD" : 9.5659,
      "HRK" : 7.5595,
      "HUF" : 357.86,
      "IDR" : 17168.2,
      "ILS" : 3.9289,
      "INR" : 90.204,
      "ISK" : 156.3,
      "JPY" : 127.03,
      "KRW" : 1339.3,
      "MXN" : 24.3543,
      "MYR" : 4.9482,
      "NOK" : 10.381,
      "NZD" : 1.6916,
      "PHP" : 59.296,
      "PLN" : 4.516,
      "RON" : 4.872,
      "RUB" : 90.8175,
      "SEK" : 10.0653,
      "SGD" : 1.6246,
      "THB" : 36.921,
      "TRY" : 9.0554,
      "USD" : 1.2338,
      "ZAR" : 18.5123
   }
}
```

Example: Water Level Service

- German Waterways Authority (Wasser- und Schifffahrtsverwaltung des Bundes)
  provides web services to query water levels and other measurement data

- The services come in a REST and a SOAP implementation
  - SOAP web services are XML-based
  - REST uses JSON
    (this is not strictly required, but most REST APIs do so)

REST URL to query for the water level of river MOSEL at KOBLENZ

https://www.pegelonline.wsv.de/webservices/rest-api/v2/stations/KOBLENZ%20UP.json?includeTimeseries=true&includeCurrentMeasurement=true

SOAP URLS

https://www.pegelonline.wsv.de/webservices/version2_4/2009/05/12/PegelonlineWebservice?WSDL

https://wsdlbrowser.com/soapclient

https://wsdlbrowser.com/soapclient?wsdl_url=https%3A%2F%2Fwww.pegelonline.wsv.de%2Fwebservices%2Fversion2_4%2F2009%2F05%2F12%2FPegelonlineWebservice%3FWSDL&function_name=getMessungenAktuell

Example SOAP Request

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/" xmlns:ns1="http://pegelonline.wsv.de/webservices/version2_4/2009/05/12">
  <SOAP-ENV:Body>
    <ns1:getMessungenAktuell>
      <ns1:parameterName>WASSERSTAND ROHDATEN</ns1:parameterName>
      <ns1:messstellenNummer></ns1:messstellenNummer>
      <ns1:messstellenName>KOBLENZ UP</ns1:messstellenName>
      <ns1:start>2020-01-30T11:00:00+01:00</ns1:start>
      <ns1:ende>2020-01-30T15:00:00+01:00</ns1:ende>
    </ns1:getMessungenAktuell>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Sadly, the linked “wsdlbrowser.com" was shut down meanwhile…
I’ll provide an example client in Java instead and demonstrate it within the lecture.

Example: University User Directory

- REST API to lookup person names and mail addresses
- HTTPS server-side certificate, with basic client authentication
- GET for search
  - `curl -u riediger@uni-koblenz.de “https://ist.uni-koblenz.de/dl/lookup/persons/…”`
  - `curl -u riediger@uni-koblenz.de “https://ist.uni-koblenz.de/dl/lookup/persons/?name=…”`

    … = (part of) lastname, firstname, mail
- GET for individual lookup
  - `curl -u riediger@uni-koblenz.de “https://ist.uni-koblenz.de/dl/lookup/person/…”`

    … = complete university mail address
- Response formats JSON or XML

What we have learned…

Communication (Part I + II)

- ✓ Network basics
- ✓ HTTP
- ✓ Sessions
- ✓ Scaling and Load Balancing

- ✓ Web Services
- ✓ SOAP and REST
- ✓ Data Representation at Layer Boundaries
- ✓ Web Service Examples

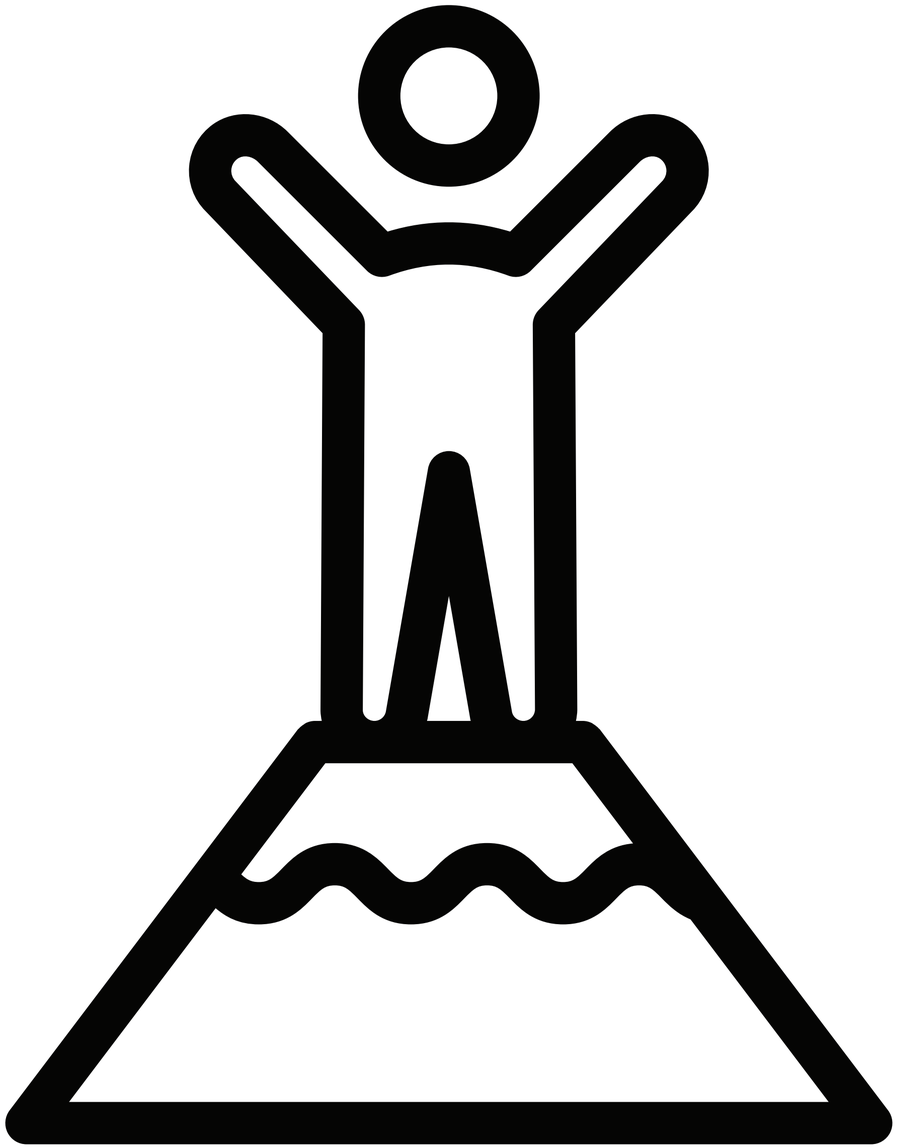


Image: colourbox.de