

# Middleware and Web Services

## Lecture 8: SOAP and REST

**doc. Ing. Tomáš Vitvar, Ph.D.**

tomas@vitvar.com • @TomasVitvar • <http://vitvar.com>



Czech Technical University in Prague

Faculty of Information Technologies • Software and Web Engineering • <http://vitvar.com/courses/mdw>



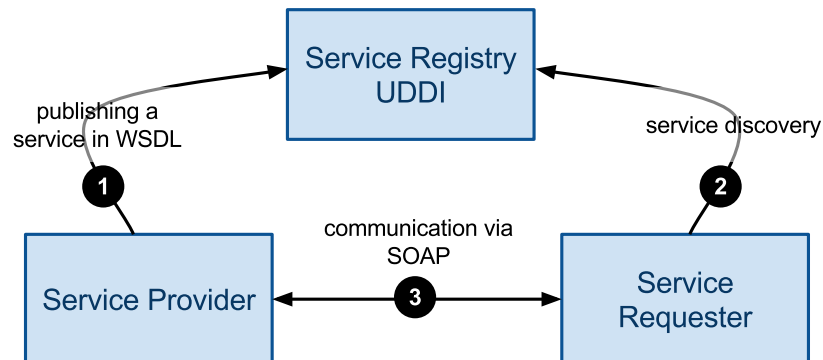
Modified: Sun Jan 04 2015, 19:18:56  
Humla v0.3

## Overview

- **SOAP**
  - *Message Path*
- Representational State Transfer
- SOAP and REST Comparison

# Web Service Architecture

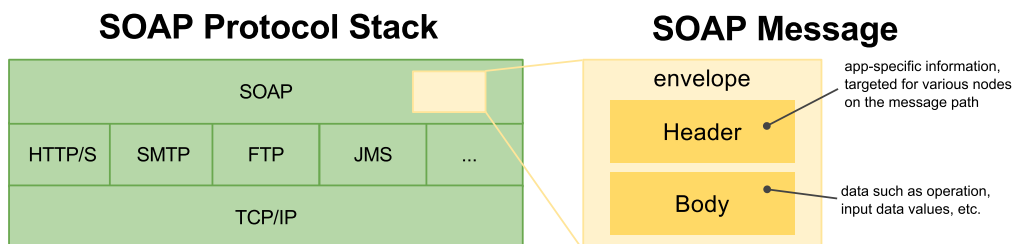
- WSDL, SOAP and UDDI



- *Realization of SOA*
- *Message-Oriented view*
  - *SOAP messaging (header, body)*
  - *types of messages – input, output, fault*

# SOAP Protocol

- SOAP defines a messaging framework



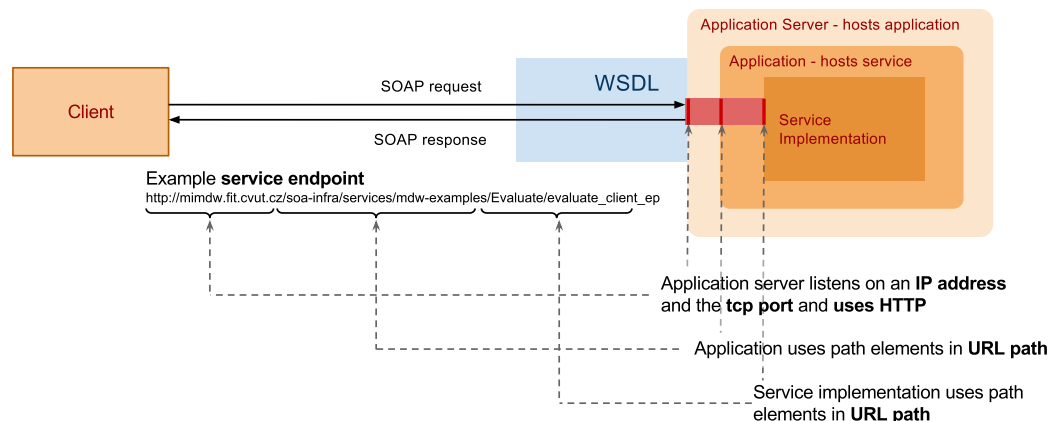
- *XML-based protocol*
- *a layer over transport protocols*
  - *binding to HTTP, SMTP, JMS, ...*
- *involves multiple nodes (message path)*
  - *sender, receiver, intermediary*

# SOAP Message

- Envelope
  - *A container of a message*
- Header
  - *Metadata – describe a message, organized in header blocks*
    - *routing information*
    - *security measures implemented in the message*
    - *reliability rules related to delivery of the message*
    - *context and transaction management*
    - *correlation information (request and response message relation)*
  - *WS extensions (WS-\*) utilize the message header*
- Body (payload)
  - *Actual contents of the message, XML formatted*
  - *Contains also faults for exception handling*
- Attachment
  - *Data that cannot be serialized into XML such as binary data*

# Endpoint

- SOAP service endpoint definition



- *Endpoint – a network address used for communication*
- *Communication – request-response, SOAP messages over a communication (application) protocol*
- *Synchronous communication – only service defines endpoint*
- *Asynchronous communication – service and client define endpoints*

## Service Invocation Example (1)

- Example service implementation
  - A service that evaluates an expression
  - Uses SOAP over HTTP
  - We can use standard HTTP tools to invoke the service
- SOAP request message

evaluate-input.xml

```
1 <soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
2   <soap:Body>
3     <ns1:evaluateRequest
4       xmlns:ns1="http://xmlns.oracle.com/mdw_examples/Evaluate/evalu
5         <ns1:x>12</ns1:x>
6         <ns1:y>18</ns1:y>
7       </ns1:evaluateRequest>
8     </soap:Body>
9 </soap:Envelope>
```

- Invoking the service using `curl`

```
1 curl -s -X POST --header "Content-Type: text/xml; charset=UTF-8" \
2 --header "SOAPAction: \"evaluate\"" --data @evaluate-input.xml \
3 http://mimdw.fit.cvut.cz/soa-infra/services/mdw-examples/Evaluate/evaluate_cli
```

## Service Invocation Example (2)

- Invocation result

```
1 * About to connect() to mimdw.fit.cvut.cz port 80 (#0)
2 * Trying 147.32.233.55... connected
3 * Connected to sb.vitvar.com (147.32.233.55) port 80 (#0)
4 > POST /soa-infra/services/mdw-examples/Evaluate/evaluate_client_ep HTTP/1.1
5 > User-Agent: curl/7.19.7 (x86_64-redhat-linux-gnu) libcurl/7.19.7 NSS/3.14.0.
6 > Host: mimdw.fit.cvut.cz
7 > Accept: */*
8 > Content-Type: text/xml; charset=UTF-8
9 > SOAPAction: "evaluate"
10 > Content-Length: 302
11 >
12 } [data not shown]
13 < HTTP/1.1 200 OK
14 < Date: Sun, 17 Nov 2013 11:24:59 GMT
15 < Server: Oracle-Application-Server-11g
16 < Content-Length: 569
17 < X-ORACLE-DMS-ECID: 004upqiWhdD0zkWVLybQ8A0005uX0004Y^
18 < SOAPAction: ""
19 < X-Powered-By: Servlet/2.5 JSP/2.1
20 < Content-Type: text/xml; charset=UTF-8
21 < Content-Language: en
```

## Service Invocation Example (3)

- SOAP response message

```
1  <?xml version="1.0"?>
2  <env:Envelope xmlns:env="http://schemas.xmlsoap.org/soap/envelope/"
3      xmlns:wsa="http://www.w3.org/2005/08/addressing">
4      <env:Header>
5          <wsa:MessageID>urn:E42018C04F7A11E3BFD5D1953058407C</wsa:MessageID>
6      </env:Header>
7      <env:Body>
8          <evaluateResponse
9              xmlns="http://xmlns.oracle.com/mdw_examples/Evaluate/evaluate">
10             <result>30</result>
11          </evaluateResponse>
12      </env:Body>
13  </env:Envelope>
```

## Client Implementation

- WSDL – Web Service Description Language
  - definitions for the client to know how to communicate with the service
    - which operations it can use
    - data formats for input (request), output (response) and fault messages
    - how to serialize the data as payloads of a communication protocol (binding)
    - where the service is physically present on the network
- Clients' environments
  - Clients implemented in a language such as Java
    - Tools to generate service API for the client, e.g. WSDL2Java
    - Can be written manually too, e.g. our example in bash
  - Clients reside on the middleware, e.g. on an Enterprise Service Bus
    - They provide added values in end-to-end communication, proxy services, SOAP intermediaries

## Overview

- SOAP
  - *Message Path*
- Representational State Transfer
- SOAP and REST Comparision

## SOAP Node

- A program that services use to transmit SOAP messages
  - *Element of SOAP messaging infrastructure*
  - *Realized by a SOAP communication server*
    - *Every vendor has its own implementation*
    - *Must conform to SOAP standard*
    - *Sometimes causes vendor interoperability problem*
- SOAP node types
  - **SOAP sender** – *a node that transmits a message*
  - **SOAP receiver** – *a node that received a message*
  - **SOAP intermediary** – *a node that receives and transmits a message and optionally processes the message*
  - **initial SOAP sender** – *the first node transmitting the message*
  - **ultimate SOAP receiver** – *the last node receiving the message*

## Message Path

- A path from a service requester to a service consumer over a number of SOAP nodes.

$S \rightarrow (I)^* \rightarrow R$ , where

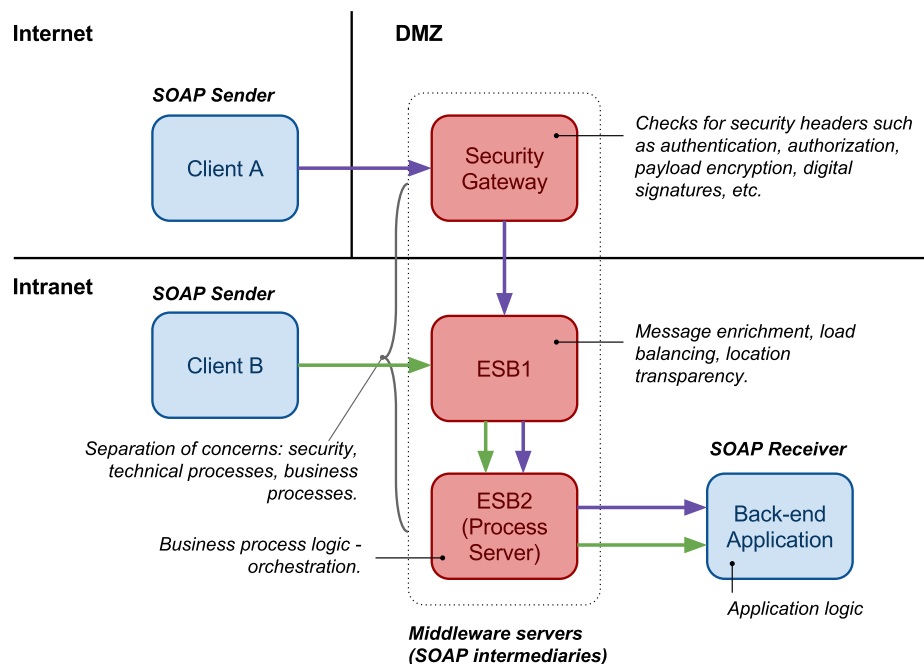
*S* is an initial sender,

*I* is an intermediary,

*R* is an ultimate receiver.

- A path may have zero or more intermediaries
  - **Passive intermediary** – such as a load balancing server
    - only forwards the message, does not rewrite headers
  - **Active intermediary** – such as a policy enforcement server
    - May modify headers

## SOAP Message Paths Example



## SOAP Message Example 1

- Client A → Security Gateway
  - Client needs to authenticate with a username and a password

```
1  <s:envelope
2    xmlns:tns="http://company.com/2011/wsd1/order"
3    xmlns:s="http://schemas.xmlsoap.org/soap/envelope/">
4    <s:header>
5      <wss:security
6        xmlns:wss="http://schemas.xmlsoap.org/ws/2002/04/secext"
7        s:mustUnderstand="1">
8        <wss:UsernameToken>
9          <wss:username>username</wsse:Username>
10         <wss:password Type="#PasswordText">pswd</wsse:Password>
11        </wss:UsernameToken>
12      </wss:security>
13    </s:header>
14    <s:body>
15      <tns:getOrderStatus xmlns:m="http://company.com/2011/schemas/order">
16        <m:orderId>2345</m:orderId>
17        <m:customerId>2234</m:customerId>
18      </tns:getOrderStatus>
19    </s:body>
20  </s:envelope>
```

## SOAP Message Example 2

- Client B → Proxy Server
  - Client directly invokes a proxy service without authentication

```
1  <s:envelope
2    xmlns:tns="http://company.com/2011/wsd1/order"
3    xmlns:s="http://schemas.xmlsoap.org/soap/envelope/">
4    <s:header/>
5    <s:body>
6      <tns:getOrderStatus xmlns:m="http://company.com/2011/schemas/order">
7        <m:orderId>2345</m:orderId>
8        <m:customerId>2234</m:customerId>
9      </tns:getOrderStatus>
10    </s:body>
11  </s:envelope>
```



## Overview

- SOAP
- **Representational State Transfer**
- SOAP and REST Comparison

## REST

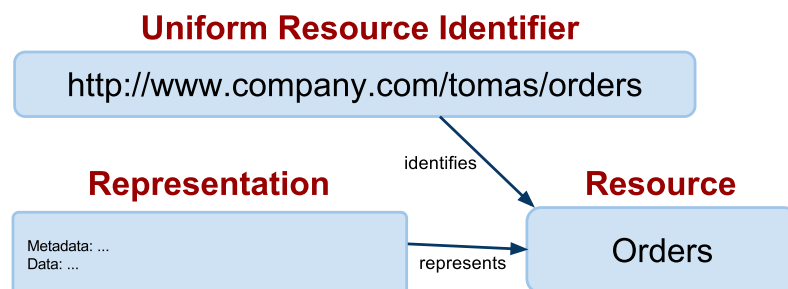
- REST
  - *Representational State Transfer*
- Architecture Style
  - Roy Fielding – co-author of HTTP
  - He coined REST in his PhD thesis [🔗](#).
    - The thesis abstracts from HTTP technical details
    - HTTP is one of the REST implementation → **RESTful**
    - REST is a leading programming model for Web APIs
- REST (RESTful) proper design
  - people break principles often
  - See *REST Anti-Patterns* [🔗](#) for some details.
- REST and Web Service Architecture
  - REST is a realization of WSA resource-oriented model

## REST Core Principles

- REST architectural style defines constraints
  - *if you follow them, they help you to achieve a good design, interoperability and scalability.*
- Constraints
  - *Client/Server*
  - *Statelessness*
  - *Cacheability*
  - *Layered system*
  - *Uniform interface*
- Guiding principles
  - *Identification of resources*
  - *Representations of resources and self-descriptive messages*
  - *Hypermedia as the engine of application state (HATEOAS)*

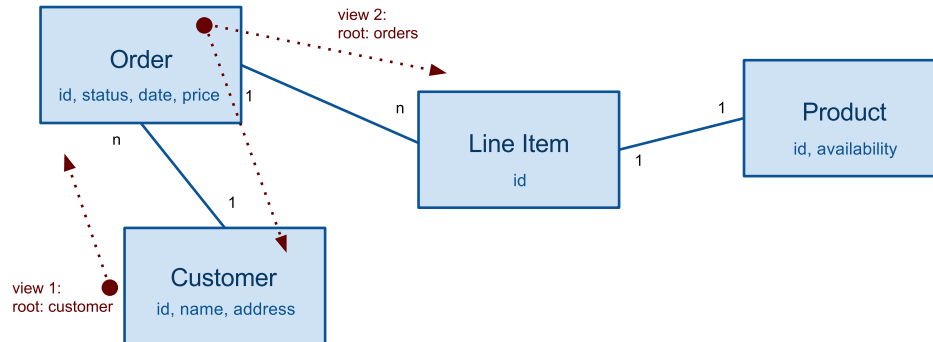
## Resource

- A resource can be anything such as
  - *A real object: car, dog, Web page, printed document*
  - *An abstract thing such as address, name, etc. → RDF*
- A resource in REST
  - *A resource corresponds to one or more entities of a data model*
  - *A representation of a resource can be conveyed in a message electronically (information resource)*
  - *A resource has an identifier (URI) and a representation (XML, JSON, ...) and a client can apply an access to it (use HTTP methods)*



## Resources over Entities

- Application's data model
  - *Entities and properties that the app uses for its data*



- URI identifies a resource within the app's data model
  - **path** – a "view" on the data model
    - data model is a graph
    - URI identifies a resource using a path in a tree with some root

## Examples of Views

- View 1
  - all customers: **/customers**
  - a particular customer: **/customers/{customer-id}**
  - All orders of a customer: **/customers/{customer-id}/orders**
  - A particular order: **/customers/{customer-id}/orders/{order-id}**
- View 2
  - all orders: **/orders**
  - All orders of a customer: **/orders/{customer-id}**
  - A particular order: **/orders/{customer-id}/{order-id}**
- Various views represented by **URL path**

## Uniform Interface

- Finite set of operations
  - They are not dependent on the domain semantics
  - They only define how to manipulate with resources
- RESTful service – HTTP methods
  - GET – reads a resource (+ HEAD, OPTIONS)
  - PUT – updates or creates a resource (+ PATCH)
  - POST – creates a new resource
  - DELETE – deletes a resource
- HTTP methods' properties
  - a method is **safe**
    - It does not change the application state (it does not modify the data)
    - GET, OPTIONS, HEAD
    - Results can be cached by intermediaries (e.g. proxy servers)
  - a method is **idempotent**
    - Every method invocation will always have the same effect
    - GET, PUT, DELETE

## Examples

- Operation **getCustomerOrder(customerId, orderId)**
  - Retrieves a representation of the order resource that belongs to a particular customer

```
1 | > GET /customers/{customerId}/orders/{orderId}
```
- Operation **openOrder(customerId)**
  - Creates a new order for a customer

```
1 | > POST /customers/{customerId}/orders
2 | < Location: /customers/{customerId}/orders/{orderId}
3 |
4 | > GET /customers/{customerId}/orders/{orderId}
```
- Operation **addLineItem(customerId, orderId)**
  - Adds a new item to the order

```
1 | > POST /customers/{customerId}/orders/{orderId}
2 | < Location: /customers/{customerId}/orders/{orderId}/items/{itemId}
3 |
4 | > GET /customers/{customerId}/orders/{orderId}/items/{itemId}
```
- Operation **closeOrder(customerId, orderId)**
  - Closes the order (i.e., changes a state of the order resource)

```
1 | > PUT /customers/{customerId}/orders/{orderId}
2 | > <status>CLOSED</status>
```

## Examples – evaluate operation

- Example REST implementation of the SOAP service **evaluate**
- Operation **evaluate(n1, n2)**
  - *Evaluates expression such that the result is **n1+n2***

```
1 > POST /evaluate/additions/  
2 > <parameters>  
3 > <n1>{n1}</n1>  
4 > <n2>{n2}</n2>  
5 > </parameters>  
6 < Location: /evaluations/additions/{n1}+{n2}  
7  
8 > GET /evaluations/additions/{n1}+{n2}
```

## Overview

- SOAP
- Representational State Transfer
- **SOAP and REST Comparision**

## SOAP vs. REST

- SOAP uses input and output messages in operations
- REST uses resources and defines access on them
- SOAP can use more protocols
- REST uses HTTP
  - *Practically, most of the SOAP implementations use SOAP over HTTP*
- Operations in SOAP are domain-specific
- HTTP operations are independent on domain semantics
  - *REST operations' semantics is defined by HTTP method + resource semantics*
- SOAP uses XML and XML Schema
- REST can use many representation formats
  - *For example, XML, JSON, YAML, etc.*
- SOAP is defined by WSDL
- REST is described in text or HTML
- Client libraries can be generated from WSDL
- REST vendor provides client libraries

## SOAP vs. REST

- SOAP clients must hard-code service's public process
- REST clients can follow links in hypertext for application states
- SOAP services are used for inter/intra-enterprise integration
- REST services are used for Web APIs for integration on the Web