

# Middleware and Web Services

## Lecture 7: Service Concepts and Technologies

**doc. Ing. Tomáš Vitvar, Ph.D.**

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



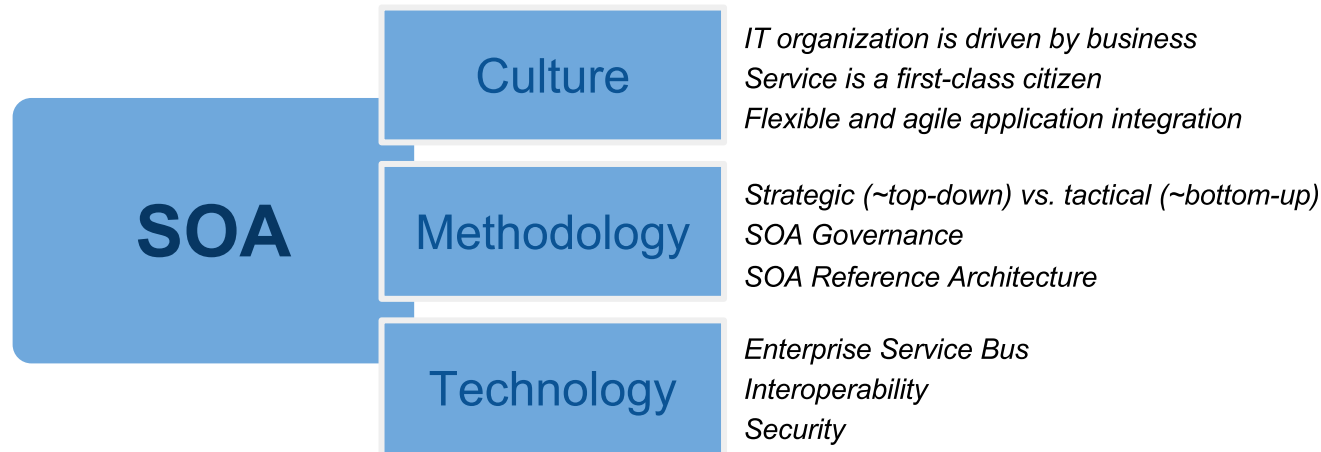
Czech Technical University in Prague

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



Modified: Sun Nov 26 2017, 23:11:12  
Humla v0.3

# Service Oriented Architecture



- SOA supports two core business strategies
  - *Growing top-line revenue*
    - *Enterprise reacts quickly to requirements from the market*
    - *Business processes can be reconfigured rather than reimplemented*
  - *Improving bottom-line profit*
    - *Saving development costs by resuing existing services*
- Pre-integrated solutions
  - *Out-of-the-box applications and integration solutions among them*

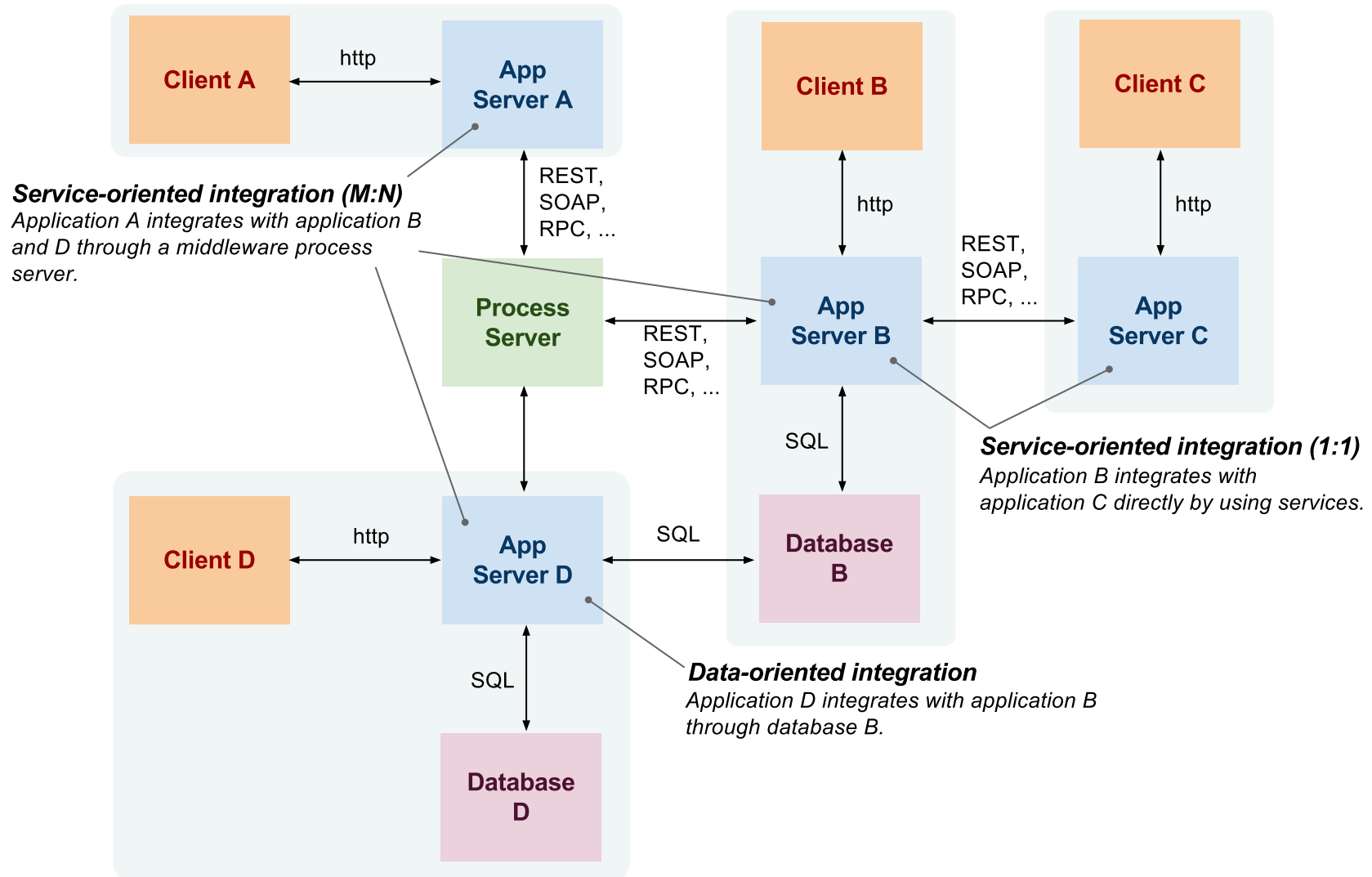
# Overview

- Integrating Applications
- Web Service Architecture
- Web Service Technologies

# Integration and Interoperability

- Integration
  - *A process of connecting applications so that they can exchange and share capabilities, that is — information and functionalities.*
  - *Includes methodological approaches as well as technologies*
- Interoperability
  - *Ability of two or more applications to understand each other*
  - *Interoperability levels*
    - *Data – syntax/structure and semantics*
    - *Functions/Processes – syntax and semantics*
    - *Technical aspects – protocols, network addresses, etc.*

# Integration Approaches Overview

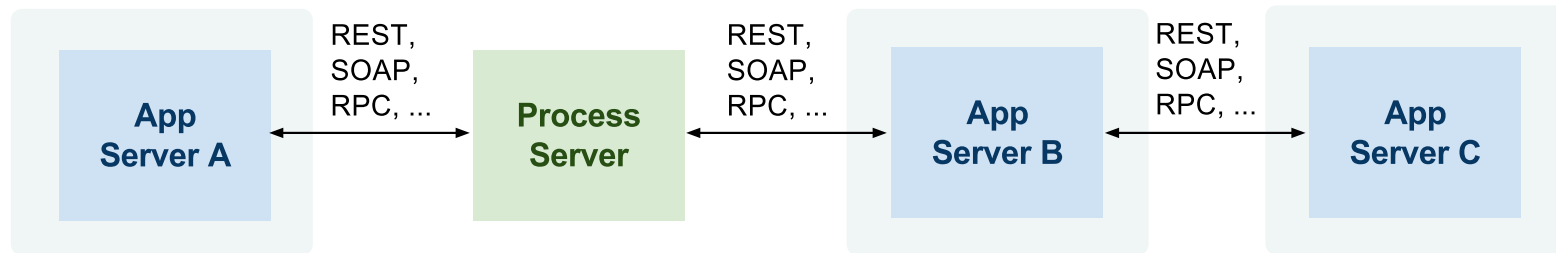


# Data-oriented Integration



- Third-party database access
  - *Application D accesses a database of application B directly by using SQL and a knowledge of database B structure and constraints*
  - *In the past: monolithic and two-tier client/server architectures*
  - *Today: ETL (Extract, Transform, Load) technologies*
- Problems
  - *App D must understand complex structures and constraints*
    - *Data – very complex, includes structure and integrity constraints*
    - *Functions/processes – hidden in integrity constraints*
    - *Technical – access mechanisms can vary*

# Service-oriented Integration



- Integration at the application layer
  - *Application exposes services that other applications consume*
  - *Services hide implementation details but only define interfaces for integration*
- Problems
  - *Can become unmanageable if not properly designed*
  - *Interoperability*
    - *Data – limited to input and output messages only*
    - *Functions/processes – limited to semantics of services*
    - *Technical – access mechanisms can vary*

# One-to-One Service Integration

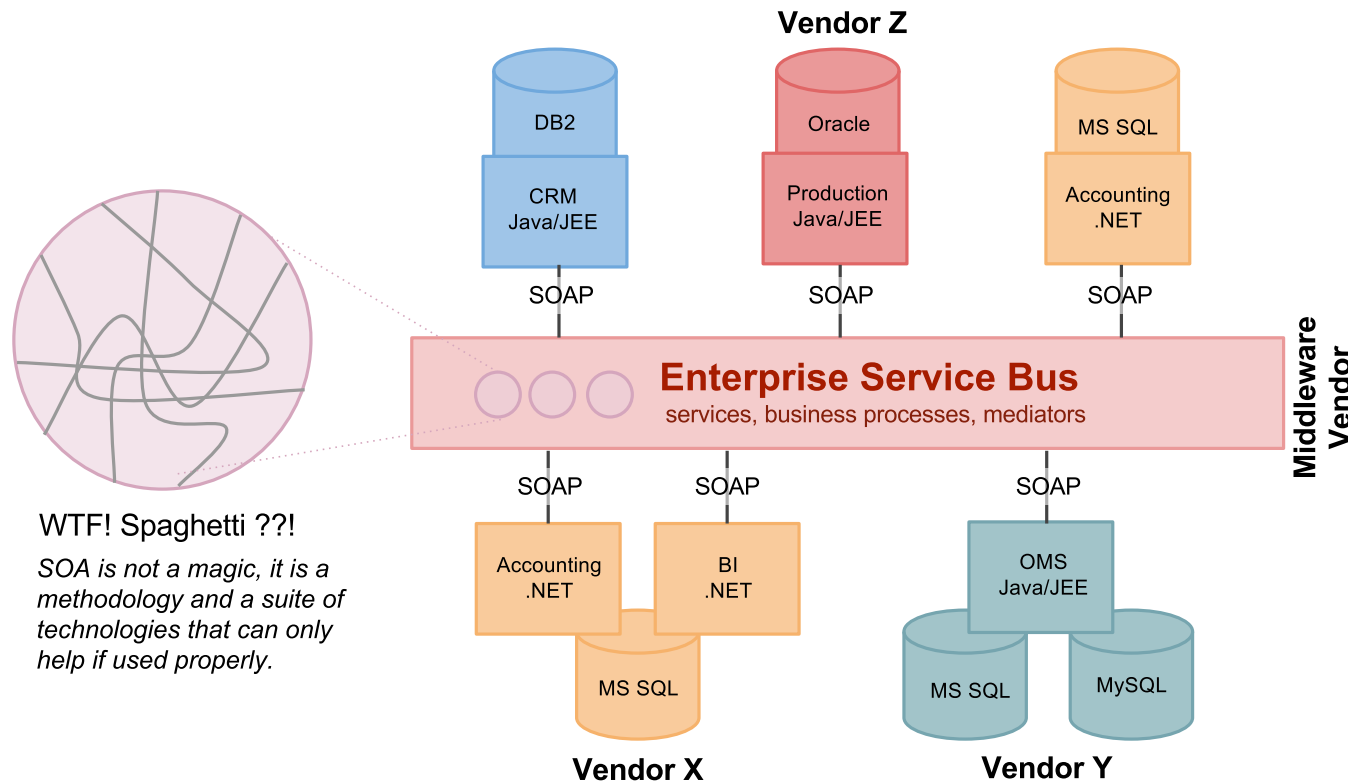
- Direct integration of applications
  - *Multiple protocols problem, multiple vendor problem*
  - *Replication of integration functionalities such as interoperability solutions*





# Many-to-Many Service Integration

- Enterprise Service Bus – central integration technology
  - *Realizes so called Service Oriented Architecture (SOA)*
  - *Contains various integration components such as process server, mediators, messaging middleware, identity management, etc.*



# Integration and Types of Data

- Transactional data – Web services
  - *Service-oriented integration*
  - *online, realtime communication between a client and a service*
  - *Usually small amount of data and small amount of service invocation in a process*
- Bulk data – ETL
  - *Data-oriented integration*
  - *processing of large amount of data in batches*
- **ESB provides both Web service and ETL capabilities**

# Overview

- Integrating Applications
- **Web Service Architecture**
- Web Service Technologies

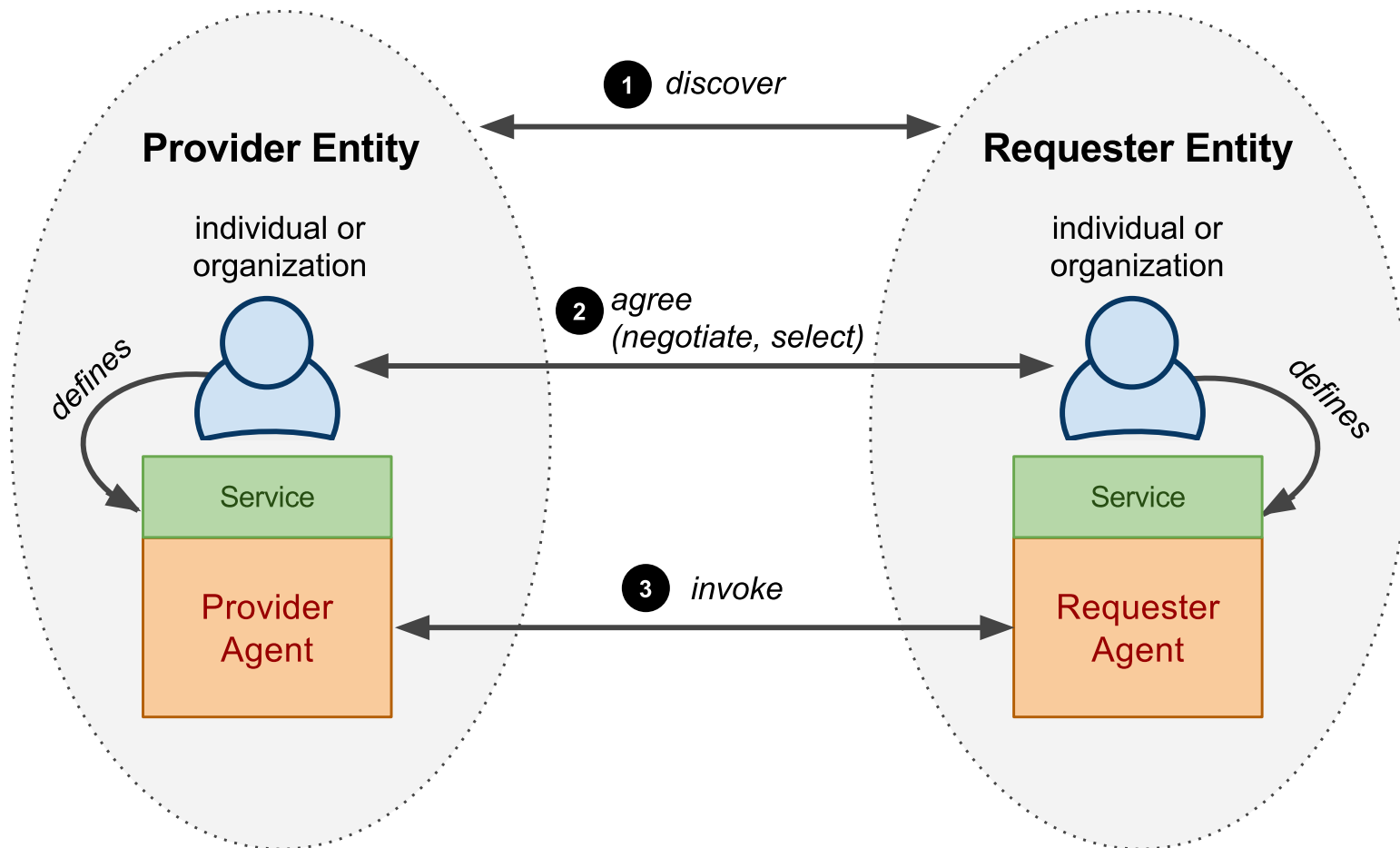
# Web Service Architecture

- Web Service Architecture
    - Defined by W3C in Web Service Architecture Working Group Note [🔗](#)
    - Defines **views**
      - message-oriented view (WSDL and SOAP)
      - resource-oriented view (REST and HTTP)
    - Defines **architecture entities** and their **interactions**
      - Abstraction over underlying technology
      - Basis for service usage processes and description languages
  - Service Oriented Architecture
    - Collection of tools, methods and technologies
    - There is some implicit understanding of SOA in the community such as
      - SOA provides advances over Enterprise Application Integration
      - SOA is realized by using SOAP, WSDL, (and UDDI) technologies
      - SOA utilizes Enterprise Service Bus (ESB)
- ⇒ ~ a realization of Web Service Architecture message-oriented view

# Basic Entities

- Agent
  - *software or hardware that sends/receives messages*
  - *concrete implementation of a service*
- Service
  - *abstract set of functionality and behavior*
  - *two different agents may realize the same service*
- Provider
  - *owner (person or organization) that provides an agent realizing a service*
  - *also called a service provider*
- Requester
  - *a person or organization that wishes to make use of a provider's service*
  - *uses a requester's agent to exchange messages with provider's agent*

# Interaction of Entities



# Service

- Difficult to agree on one definition
- Business definition
  - *A service realizes an effect that brings a business value to a service consumer*  
→ *for example, to pay for and deliver a book*
- Conceptual definition
  - *service characteristics*  
→ *encapsulation, reusability, loose coupling, contracting, abstraction, discoverability, composability*
- Logical definition
  - *service interface, description and implementation*
  - *service usage process*  
→ *service use tasks, service types*
- Architectural definition
  - *business service (also application service)*  
→ *external, exposed functionality of an application*
  - *infrastructure service*  
→ *internal/technical, supports processing of requests*

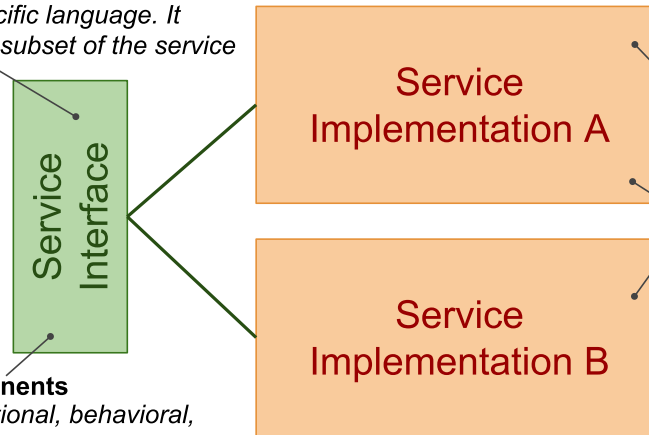
# Interface, Description and Implementation

## Service Description

An explicit description of the service interface in a specific language. It describes all or a subset of the service interface.

## Interface components

Information, functional, behavioral, non-functional, technical



## Implementation Technology

Service implementation implements the service interface in a specific implementation environment.

## Multiple service implementations

One service interface can be realized by many different service implementations.

- Terminology clarification
  - *service ~ service interface + service implementation*
  - *WSDL service ~ service description in WSDL language*
  - *SOAP service ~ a service interface is possible to access through SOAP protocol; there is a WSDL description usually available too.*
  - *REST/RESTful service ~ service interface that conforms to REST architectural style and HTTP protocol*



# Service Characteristics

## Loose Coupling

The requester agent's implementation is independent from service usage. That is, there is no "hard-wired" knowledge required to use the service.

## Reusability

The service can be used in many different scenarios by different requester agents that are unforeseen during the service design.

## Contracting

The service interface is a contract between the requester and the provider. They both agree to follow the service description in order to achieve interoperability.

## Abstraction

Service interface is abstracted from underlying service implementation as well as all software and hardware technology.

## Discoverability

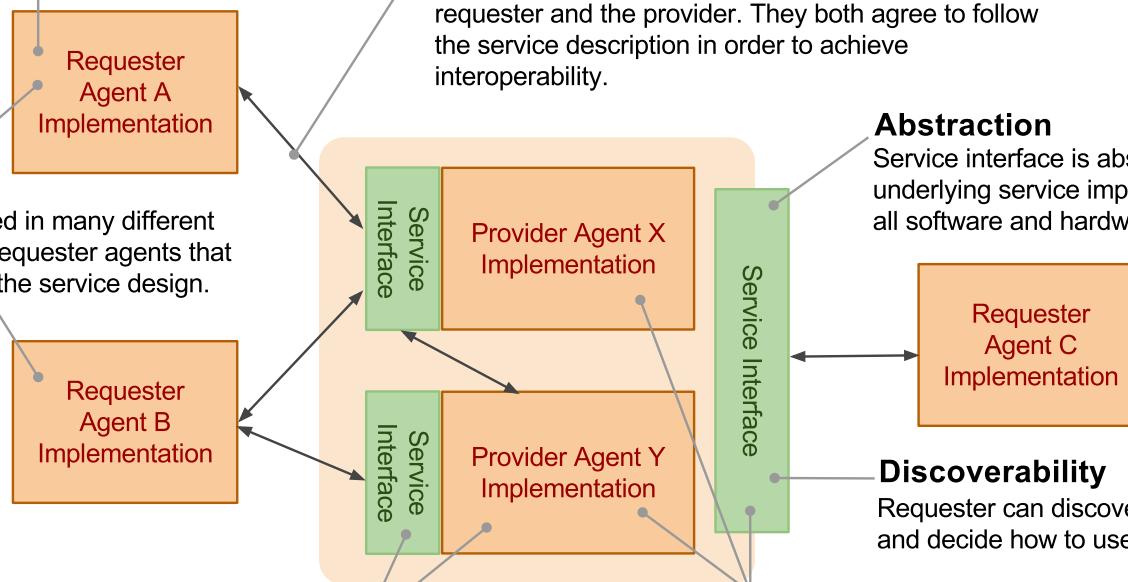
Requester can discover the service interface and decide how to use it.

## Encapsulation

The provider agent implementation is hidden to the requester agent accessing the service. The requester agent only knows the service interface to consume its functionality.

## Composability

It is possible to compose services into more complex processes. Such processes can again be accessed as services.

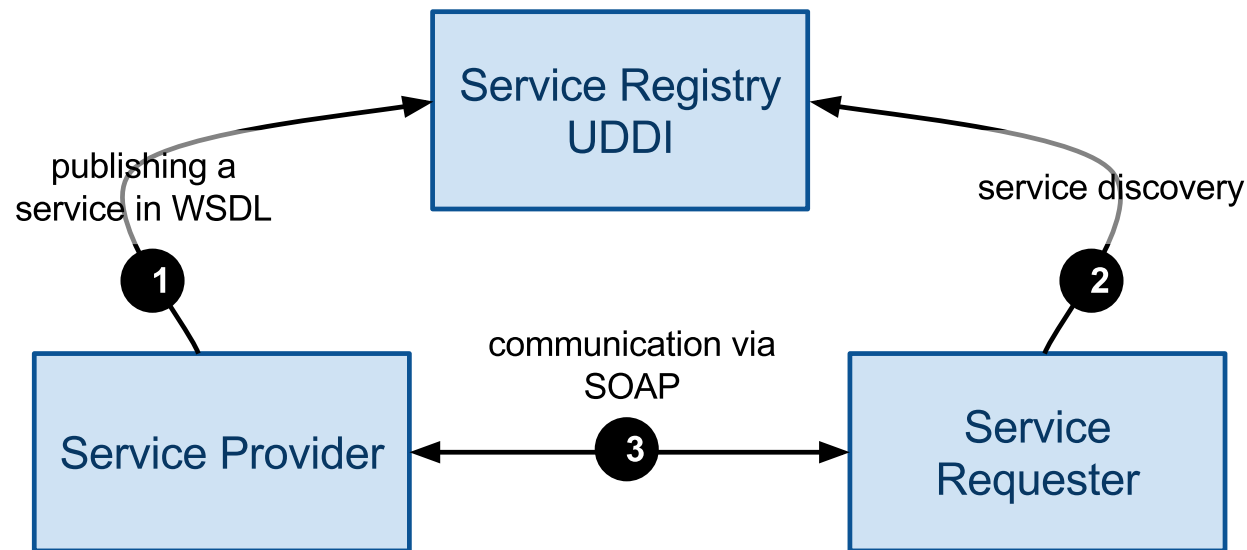


# Overview

- Integrating Applications
- Web Service Architecture
- Web Service Technologies
  - *SOAP*
  - *WSDL*
  - *WS-Addressing*
  - *REST*
  - *Comparision*

# Web Service Architecture

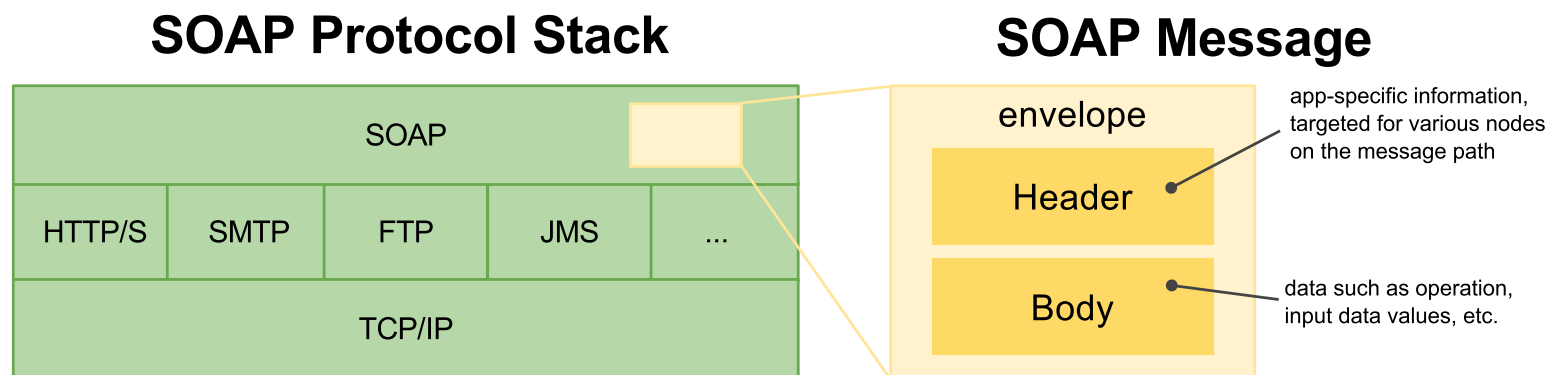
- WSDL, SOAP and UDDI



- *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/evaluate"
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_client_
```

# 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/3.14.0.0 z1
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

- Integrating Applications
- Web Service Architecture
- Web Service Technologies
  - *SOAP*
  - *WSDL*
  - *WS-Addressing*
  - *REST*
  - *Comparision*

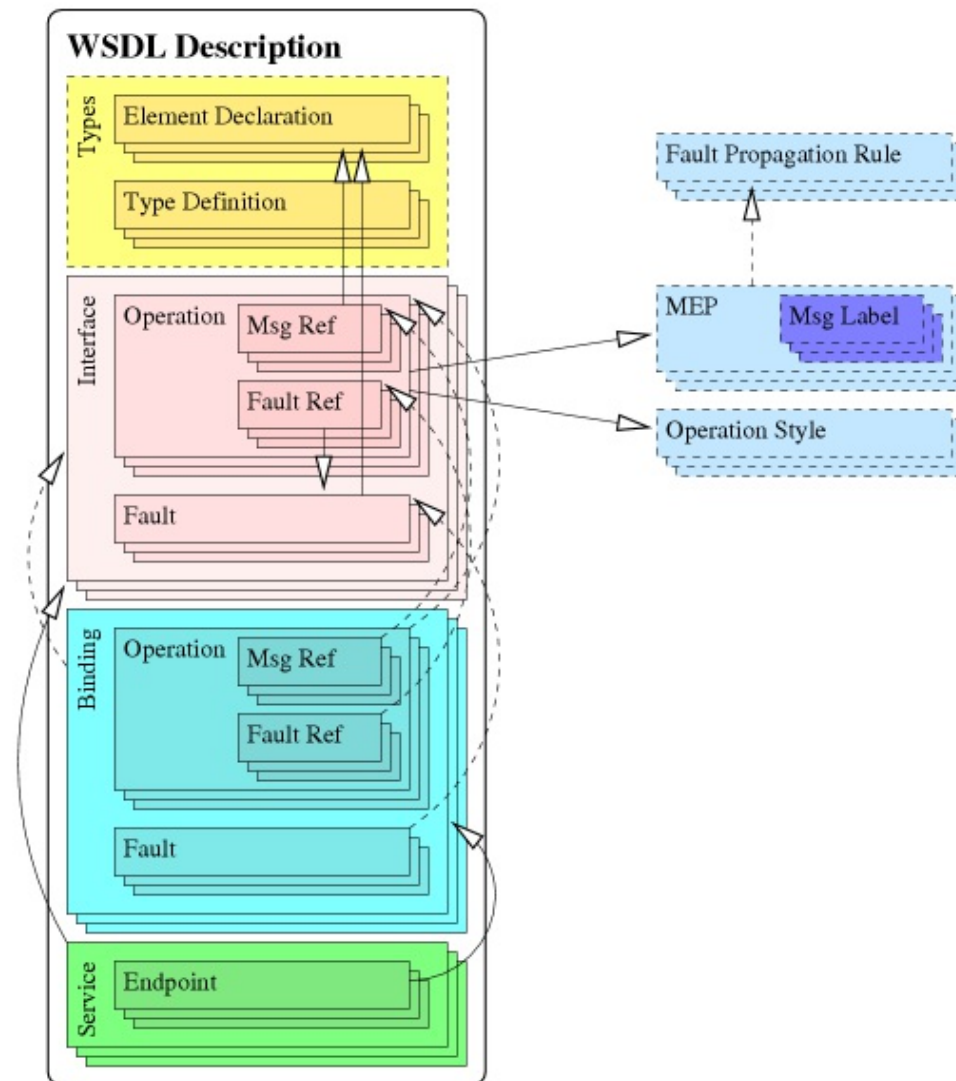
# Specifications

- WSDL = Web Service Description Language
  - *A standard that allows to describe Web services explicitly (main aspects)*
  - *A contract between a requester and a provider*
- Specifications
  - *WSDL 1.1 – still widely used*
    - *Web Service Description Language 1.1* [!\[\]\(a22ba4e13c745edbf29e51af246c4c12\_img.jpg\)](#)
  - *WSDL 2.0 – An attempt to address several issues with WSDL 1.1*
    - *SOAP vs. REST, naming, expressivity*
    - *WSDL 2.0 Primer (part 0)* [!\[\]\(33b18af9a4b997eb52666cfeb3c44157\_img.jpg\)](#)
    - *WSDL 2.0 Core Language (part 1)* [!\[\]\(262b158440b847a82f89a14cab8644ec\_img.jpg\)](#)

# WSDL Overview and WSDL 1.1 Syntax

- Components of WSDL
  - Information model (**types**)
    - Element types, message declarations (XML Schema)
  - Set of operations (**portType**)
    - A set of operations is "interface" in the WSDL terminology
    - operation name, input, output, fault
  - Binding (**binding**)
    - How messages are transferred over the network using a concrete transport protocol
    - Transport protocols: HTTP, SMTP, FTP, JMS, ...
  - Endpoint (**service**)
    - Where the service is physically present on the network
- Types of WSDL documents
  - **Abstract WSDL** – only information model and a set of operations
  - **Concrete WSDL** – everything, a concrete service available in the environment

# WSDL Components and Dependencies



# Overview

- Integrating Applications
- Web Service Architecture
- Web Service Technologies
  - *SOAP*
  - *WSDL*
  - *WS-Addressing*
  - *REST*
  - *Comparision*

# Overview

- WS-Addressing
  - *W3C Recommendation, May 2006* [🔗](#)
  - *A transport-independent mechanisms for web services to communicate addressing information*
  - *WSDL describes WS-Addressing as a policy attached to a WSDL binding*

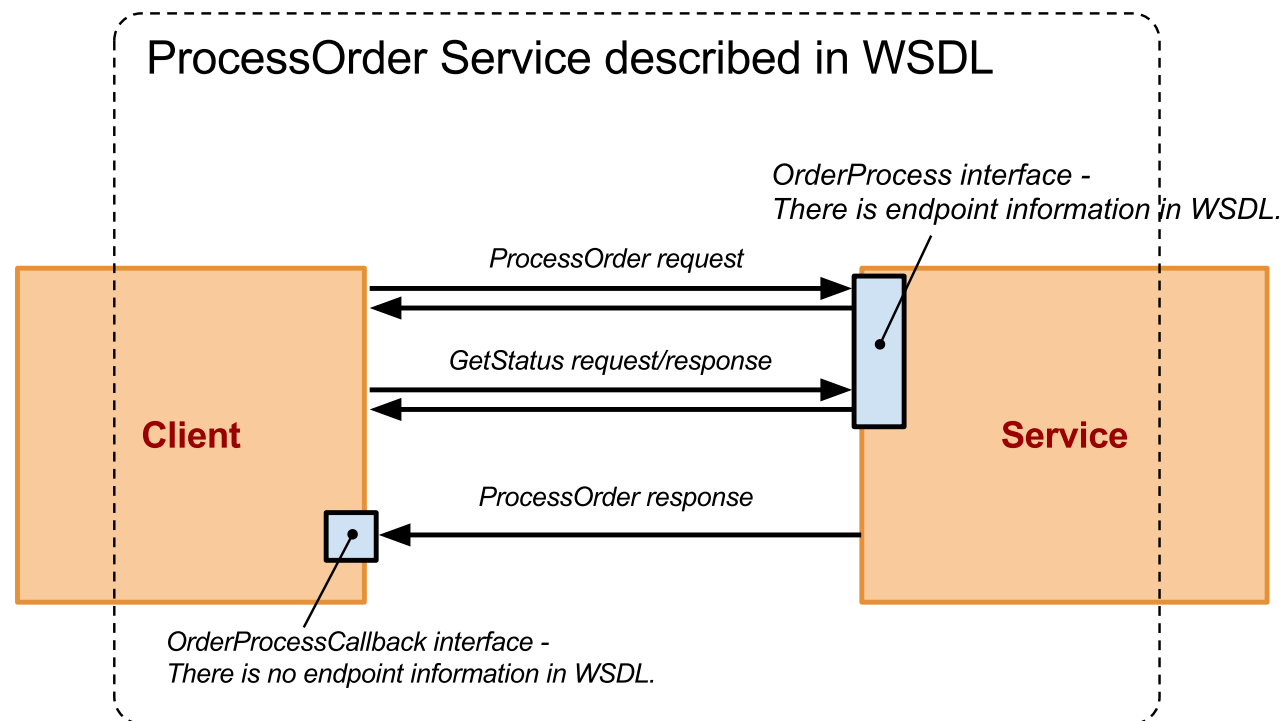
```
1 <binding name="OrderProcessBinding" type="op:OrderProcess">
2   <soap:binding transport="http://schemas.xmlsoap.org/soap/http"/>
3   <PolicyReference xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy"
4     URI="#wsaddr_policy" wsdl:required="false"/>
```

- Two main purposes
  1. *Asynchronous communication*
    - *Client sends an endpoint where the server should send a response asynchronously*
  2. *Relating interactions to a conversation*
    - *Client and service communicate conversation ID*



# Order Processing Example

- Asynchronous communication via callback, steps:
  - *Client submits an order request*
  - *Service starts processing of the order (CRM, OMS, back-office)*
  - *Client can retrieve the order status*
  - *Service responds asynchronously with an order response message*



# Interface Example (1)

- Order process complex conversation
  1. The client invokes **processOrder**.
  2. The service responds back **synchronously** with order status.
  3. The client gets the status of order processing by invoking synchronous **getStatus** operation (this can be invoked several times).
  4. The service responds back **asynchronously** by invoking **processOrderResponse** – callback on client's interface
- Interface implemented by the order process service
  - **getStatus** operation must be executed in the same **conversation** as **processOrder** operation

```
1 <portType name="OrderProcess">
2   <operation name="processOrder">
3     <input message="op:OrderProcessRequestMessage"/>
4     <output message="op:OrderStatusResponseMessage"/>
5   </operation>
6   <operation name="getStatus">
7     <input message="op:OrderStatusRequestMessage"/>
8     <output message="op:OrderStatusResponseMessage"/>
```

# Interface Example (2)

- Interface implemented by the client

```
1 <portType name="OrderProcessCallback">
2   <operation name="processOrderResponse">
3     <input message="op:OrderProcessResponseMessage"/>
4     <fault message="op:OrderProcessFaultMessage"/>
5   </operation>
6 </portType>
```

# ProcessOrder Request Message

- Client sends process order request – **processOrder**
  - it sends addressing information where the client listens for the callback
  - it sends conversation ID (message ID) to start the conversation on the server

```
1 > POST /soa-infra/services/mdw-examples/ProcessOrder/orderprocess_client_ep HTTP/1.1
2 > Host: mimdw.fit.cvut.cz
3 > Content-Type: text/xml;charset=UTF-8
4 > SOAPAction: "processOrder"
5 > Content-Length: 810
6
7 <soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
8   xmlns:ord="http://mimdw.fit.cvut.cz/mdw-examples/cdm/order">
9   <soap:Header xmlns:wsa='http://www.w3.org/2005/08/addressing'>
10     <wsa:ReplyTo>
11       <wsa:Address>http://192.168.94.110:2233/path/to/service</wsa:Address>
12     </wsa:ReplyTo>
13     <wsa:MessageID>urn:AXYYBA00531111E3BFACA780A7E5AF64</wsa:MessageID>
14   </soap:Header>
15   <soap:Body>
16     <ord:Order>
17       <ord:CustomerId>1</ord:CustomerId>
18       <ord:LineItems>
19         <ord:item>
20           <ord:label>Apple MacBook Pro</ord:label>
21           <ord:action>ADD</ord:action>
22         </ord:item>
23       </ord:LineItems>
24     </ord:Order>
25   </soap:Body>
26 </soap:Envelope>
```

# GetStatus Request Message

- Client sends get status request – **getStatus**
  - after it invokes **processOrder** with conversation ID (message ID)
  - it uses the same conversation ID for get status request too
    - the request will be processed by the running service instance

```
1 > POST /soa-infra/services/mdw-examples/ProcessOrder/orderprocess_client_ep HTTP/1.1
2 > Host: mimdw.fit.cvut.cz
3 > Content-Type: text/xml; charset=UTF-8
4 > SOAPAction: "getStatus"
5 > Content-Length: 472
6
7 <soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
8   <soap:Header xmlns:wsa="http://www.w3.org/2005/08/addressing">
9     <wsa:RelatesTo>urn:AXYYBA00531111E3BFACA780A7E5AF64</wsa:RelatesTo>
10   </soap:Header>
11   <soap:Body>
12     <ns1:StatusRequest
13       xmlns:ns1="http://mimdw.fit.cvut.cz/mdw_examples/ProcessOrder/OrderProcess
14     <ns1:process-id>18a9baec2d5ac0a2:64d155de:1425c4185f1:-7ff2</ns1:process-i
15     </ns1:StatusRequest>
16   </soap:Body>
17 </soap:Envelope>
```

# Overview

- Integrating Applications
- Web Service Architecture
- Web Service Technologies
  - *SOAP*
  - *WSDL*
  - *WS-Addressing*
  - *REST*
  - *Comparision*

# 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

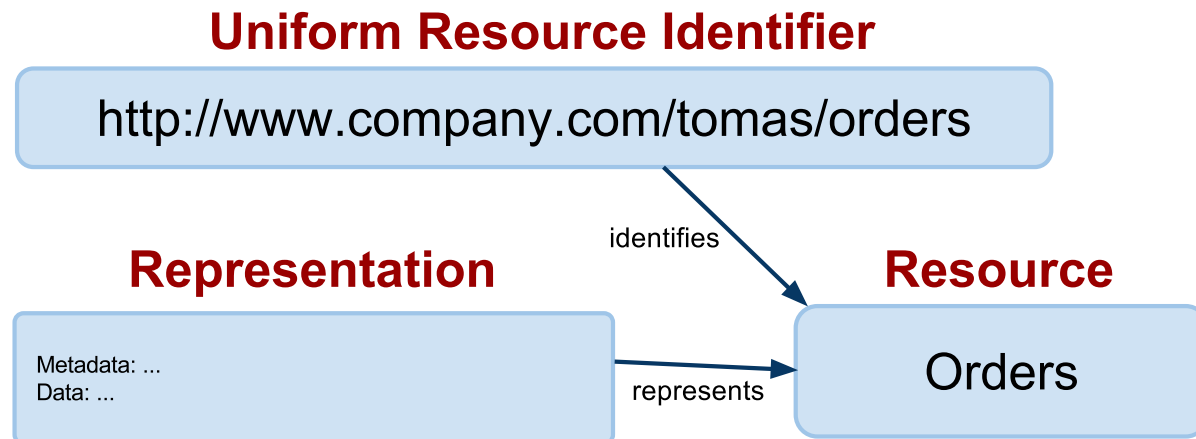
# REST Core Principles

- 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)*



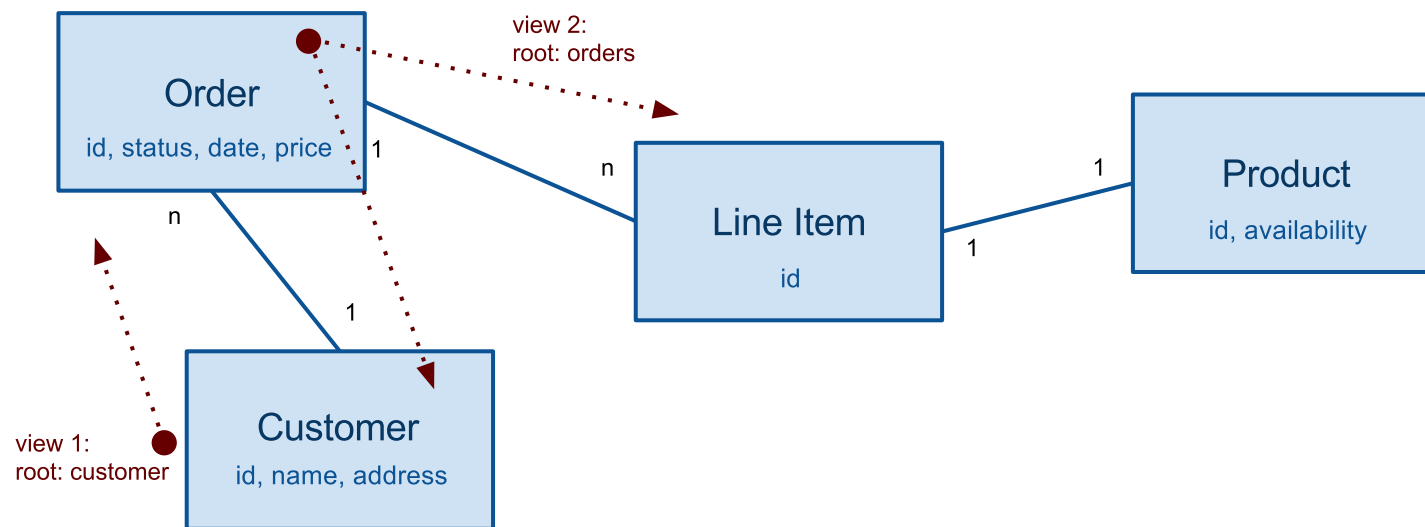
# Resource

- 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

- Integrating Applications
- Web Service Architecture
- Web Service Technologies
  - *SOAP*
  - *WSDL*
  - *WS-Addressing*
  - *REST*
  - *Comparision*

# Service Description

- Standards-driven
  - *Standards that define service description*
  - *They give a space for variability*
    - *too much – big flexibility but increases complexity (~WSDL)*
    - *limited – enforce agreement and interoperability (~REST)*  
*(as long as parties correctly implement the standard)*
- Languages to describe service interfaces
  - *formal – machine processable*
  - *textual – natural text description*
- Comparison of WSDL and REST models for service interfaces

Model	Standards-driven	Languages
<b>WSDL</b>	XML-based WSDL, XML Schema for input/output/fault messages; big space for variations (operations, exchange patterns, protocols)	WSDL+XML, textual description for rules of public processes
<b>REST</b>	Web Architecture, HTTP, XML Schema, JSON; little space for variations (uniform interface, statelessness, etc.)	HTML – mostly textual description, AtomPub, WADL



# 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