Introduction to Web Services

Held as part of the lecture series on
Web Engineering at Vienna University of Technology

May 2014
Web Engineering

Web Services

Philipp Liegl
Business Informatics Group
Institute of Software Technology and Interactive Systems
Vienna University of Technology
Favoritenstraße 9-11/188-3, 1040 Vienna, Austria
phone: +43 (1) 58801-18804 (secretary), fax: +43 (1) 58801-18896
office@big.tuwien.ac.at, www.big.tuwien.ac.at
Outline of today’s talk

- Introduction to Service-Oriented Computing
- SOAP
- WSDL
- UDDI
- Java API for XML Web Services (JAX-WS)
- RESTful Web Services
- Java API for RESTful Web Services (JAX-RS)
Introduction to Service-Oriented Computing

Context of Web Services: Distributed Information Systems

- Layers of an information system
  - Presentation layer
    - Communication interface to external entities
    - Graphical user interface for human users or non graphical user interface for other programs
  - Application logic layer
    - Implements operations requested by clients through the presentation layer
  - Resource management layer
    - Deals with different data sources of an information system
- Distributed systems are split up into parts
  - Run simultaneously on multiple computers
  - Communicate over a network
Machine to machine communication
Getting applications to talk to each other

Another information system

client

presentation layer

application logic layer

resource management layer

TUWIEN WE Information System

client

presentation layer

application logic layer

resource management layer
What is a service?
Introduction to Service-Oriented Computing

Heterogeneity as the main obstacle for seamless communication of distributed information systems

- Example: Yield management in the airline industry requires close system interaction in order to retrieve the most current prices

<table>
<thead>
<tr>
<th>Travel agency &quot;The agency&quot;</th>
<th>Airline &quot;The airline&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows Server 2008</td>
<td>UNIX System</td>
</tr>
<tr>
<td>Travel information and booking application (programmed in Java)</td>
<td>Flight information and booking application (programmed in C)</td>
</tr>
</tbody>
</table>

- Mismatch in operating system, language, platform, etc.
- **Service-oriented computing** is an emergent paradigm that helps to overcome these mismatches
Introduction to Service-Oriented Computing

Application integration using services

- **Service vs. Web Service**
  - Services are business functions which an enterprise offers to its business partners
  - A possible implementation of Services are Web Services
  - However, other concepts may also be used to implement a Service, e.g., ebXML
Introduction to Service-Oriented Computing

Important terms in service-oriented computing

- **(Web) Services** are self-contained modules that can be described, published, located, orchestrated, and programmed using XML-based technologies over a network.

- **Service providers** are organizations that provide the service implementations, supply their service descriptions, and provide related technical and business support.

- **Service clients** are end-users and organizations that use some service.

- **Service aggregators** are organizations that consolidate multiple services into a new, single orchestrated service offering that is commonly known as business process.

- **A service-oriented architecture (SOA)** is a logical way of designing a software system to provide services to either end-user applications or to other services distributed in a network, via published and discoverable interfaces.
Introduction to Service-Oriented Computing

Characteristics of Web Services (WS)

- **WS semantically encapsulate discrete functionality**
  - A Web Service is a self-contained software module that performs a single task (e.g., weather forecast by passing the zip-code as parameter)

- **WS share a contract**
  - In order to allow interaction of services, a *formal contract must be established*, that defines the exact terms of an information exchange between a service client and a service provider

- **WS abstract underlying program logic**
  - A service exposes a certain functionality to a client. How that functionality is achieved (e.g., which program language is used, or which database is used) remains invisible to the caller

- **WS are loosely coupled software modules**
  - A service interface is defined in a *neutral manner*, independent of the underlying platform, operating system, or programming language
  - Due to their neutral interfaces, services are not hard-wired. Thus, a service may be easily exchanged by another service, without much implementation effort

- **WS are reusable**
  - A service may be reused by multiple applications
Introduction to Service-Oriented Computing

Characteristics of Web Services II (WS)

• WS can be dynamically found and included in applications
  • A WS provides programmable access. Thus, a WS may be embedded in a remotely located application, i.e., a service may be composed.
  • Unlike Web Sites, Web Services are not targeted at human users
  • They are called by and exchange data with other software modules and applications.

• WS are described in terms of a standard description language
  • Web Service Description Language (WSDL) and Web Application Description Language (WADL) describe functional service characteristics
    • Functional requirements: Requirements of the functionality which must be provided. (Functions, Data, Behavior, etc.)
    • Non-functional requirements: Requirements of the circumstances under which the functionality must be provided (e.g., reliability, performance, etc.)

• WS are distributed over the Internet
  • WS make use of existing ubiquitous transport Internet protocols like HTTP
  • By relying on the same well-understood transport mechanism as Web Content, Web Services may leverage existing infrastructures and may cross corporate firewalls.
Introduction to Service Oriented Computing

Functional vs. Non-Functional Service Characteristics

• **Functional Service Characteristics**
  • Detail the operational characteristics that define the overall behavior of the service
    • How the service is invoked
    • The location where it is invoked
    • Syntax of exchanged messages, etc.
    • Typically a question of the system design

• **Non-Functional Service Characteristics**
  • Concentrate on service quality attributes
    • Service metering and cost
    • Performance metrics such as response time
    • Security attributes, authorization, authentication, transactional integrity, scalability, etc.
    • Typically a question of the system architecture
Introduction to Service Oriented Computing

Tight versus loose coupling

- Coupling refers to the degree to which software components depend upon each other.

<table>
<thead>
<tr>
<th></th>
<th>Tight coupling</th>
<th>Loose coupling</th>
<th>SOA</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Physical coupling</strong></td>
<td>Direct physical link required</td>
<td>Physical intermediary</td>
<td></td>
</tr>
<tr>
<td><strong>Communication style</strong></td>
<td>Synchronous</td>
<td>Asynchronous</td>
<td></td>
</tr>
<tr>
<td><strong>Interaction pattern</strong></td>
<td>OO-style navigation of complex object trees</td>
<td>Data-Centric, Self-Contained messages</td>
<td></td>
</tr>
<tr>
<td><strong>Control of process logic</strong></td>
<td>Central control of process logic</td>
<td>Distributed logic components</td>
<td></td>
</tr>
<tr>
<td><strong>Underlying platforms</strong></td>
<td>Homogeneous</td>
<td>Heterogeneous</td>
<td></td>
</tr>
<tr>
<td><strong>Service discovery and binding</strong></td>
<td>Statically bound services</td>
<td>Dynamically bound services</td>
<td></td>
</tr>
<tr>
<td><strong>Platform dependencies</strong></td>
<td>Strong OS and programming language dependency</td>
<td>OS- and programming language independent</td>
<td></td>
</tr>
</tbody>
</table>
Introduction to Service-Oriented Computing

Web Services

- Types of Web services
  - SOAP/WSDL based
    - Service interface is exposed through WSDL documents
    - Message exchange using SOAP
    - Client code may be generated from WSDL description
  - Representational State Transfer (REST)
    - Easy way to communicate with Web Services
    - Resources are identified by URIs and their state is manipulated through HTTP operations GET, POST, PUT, DELETE
    - Rather a set of architectural principles, than a standard
Introduction to Service-Oriented Computing

Characterizing Web Services - Integration scenarios

- Within an organization

- Between business partners
Introduction to Service-Oriented Computing

Why Web Services?

- **Interoperable**
  - connection of heterogeneous systems
  - usage of common standards

- **Economical**
  - no installation and tight integration
  - recycling of components

- **Automatic**
  - no human intervention necessary

- **Accessible**
  - access to legacy systems
  - access to internal applications

- **Available**
  - on any device, every time, anywhere

- **Goals**
  - solve integration problem
  - complete automated infrastructure for, e.g., e-commerce
Service oriented architectures are an **abstract pattern** that applies to a wide variety of Web services.

SOA is a **loosely-coupled architecture** designed to meet the business needs of an organization.

SOA represents business functions as **shared, reusable services**.

SOA defines an architecture which usually consists of the following roles and the contracts between those roles:
Introduction to Service-Oriented Computing
Operations in a Web Service Architecture

- **Publish**
  - A service description needs to be published such that
    - the service requestor can find it
    - it is accessible

- **Find**
  - The service requestor retrieves the service description
    - directly
    - by querying a service registry

- **Bind**
  - The service requestor invokes or instantiates the interaction with the service by using binding details in the service description to locate, contact, and invoke the service.
Introduction to Service-Oriented Computing

Characteristics of a SOA

- The **software components** in a SOA are **services-based** on standard protocols.

- Communication infrastructure used within a SOA should be designed to be **independent of the underlying protocol layer**.

- Share **Schema** and **Contract**, not Class/database.

- Offers coarse-grained **business services**, as opposed to fine-grained software-oriented **function calls**.

- Uses service granularity to provide effective **composition** and **management** of services.
## Introduction to Service-Oriented Computing

### Roles in a Web Service Architecture

<table>
<thead>
<tr>
<th>Role</th>
<th>Business Perspective</th>
<th>Technical Perspective</th>
</tr>
</thead>
<tbody>
<tr>
<td>service provider</td>
<td>owner of the service</td>
<td>platform that hosts the service</td>
</tr>
<tr>
<td>service requestor</td>
<td>business that requires certain functionality</td>
<td>application that invokes or interacts with the service</td>
</tr>
<tr>
<td>service registry</td>
<td>searchable registry of service descriptions where service providers publish their service descriptions</td>
<td></td>
</tr>
</tbody>
</table>
Introduction to Service-Oriented Computing

Layers in a SOA

**Business domain**

**Business processes**
- Purchasing
- Order management
- Inventory

**Business services**

**Infrastructure services**

**Component-based service realizations**

**Operational systems**
- CRM
- ERP
- Legacy Applications
- Databases
Introduction to Service-Oriented Computing

Web Service Technology Stack

Choreography – CDL4WS
Orchestration – BPEL4WS
WS-Reliability
WS-Security
Transactions
Coordination
Context
UDDI
WSDL
SOAP
XML
HTTP, JMS, SMTP

Business processes
Quality of Service
Discovery
Description
Message
Transport
Note, that finding and publishing a service is also realized using SOAP calls.
Introduction to Service-Oriented Computing

Standards of interest

- **SOAP (by W3C)**
  - Originally from “Simple Object Access Protocol“
  - Message exchange format

- **WSDL (by W3C)**
  - “Web Service Description Language“
  - Standard to describe what is exchanged between Web Service Provider and Web Service Requestor

- **UDDI (by OASIS)**
  - “Universal Description Discovery and Integration“
  - Basis for a directory service
Conventional distributed object communication protocols such as Java/RMI had a **symmetric** requirement:
- both ends of the communication link had to be implemented under the same distributed object model
- e.g., in case of Java RMI, both ends must be implemented in Java, which does the marshalling (transform Java object into the exchange format) and unmarshalling (transform exchange format back to Java object) of objects.
**SOAP**

What is SOAP?

- **SOAP** once stood for 'Simple Object Access Protocol', but this acronym was dropped with Version 1.2 of the standard.

- **SOAP** is a protocol for exchanging structured information in a decentralized, distributed environment and defines:
  - an **envelope** that defines a framework for describing what is in a message and how to process it
  - a set of **encoding rules** for expressing instances of application-defined data types
  - **conventions** for representing remote procedure calls and responses

- **XML-based W3C Specification**

- For inter-application communication

- **SOAP** describes how a message is formatted, but not how it is delivered

- A **SOAP** message must be embedded in a transport-level protocol

- Typically **HTTP** is used, however, **SMTP**, **FTP** and others may also be used.
SOAP

Web Services communication and messaging network

SOAP is a network application protocol that is used to transfer messages between service instances, described by WSDL interfaces.
SOAP
The two fundamental Web Service message exchange patterns

Sender -> Request message -> SOAP -> Receiver

Sender -> Request message -> SOAP -> Receiver -> Response message
A SOAP envelope consists of an optional header and a mandatory body.

All elements of a SOAP envelope are defined using XML Schema.

Header contains information on how the message is to be processed, e.g., routing and delivery, authentication or authorization, transaction contexts, etc.

The body element is mandatory and carries the message payload.

The content of the header and the body element are application defined and not part of the SOAP standard.
Example of a SOAP header with two header blocks

```xml
<?xml version="1.0" encoding="UTF-8"?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
  ...
  <env:Header>
    <tx:transaction-id
      xmlns:tx="http://www.transaction.com/transaction"
      env:mustUnderstand="true">
      512
    </tx:transaction-id>
    <notary:token
      xmlns:notary="http://www.notary.com/token"
      env:mustUnderstand="true">
      RKEIK-DKEKW-DKIEL-DKKLK-WIEFK
    </notary:token>
  </env:Header>
  ...
</env:Envelope>
```
Example SOAP Message

```xml
<?xml version="1.0" encoding="UTF-8"?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
    <env:Header>
        <n:alertcontrol xmlns:n="http://example.org/alertcontrol">
            <n:priority>1</n:priority>
            <n:expires>2010-06-22T14:00:00-05:00</n:expires>
        </n:alertcontrol>
    </env:Header>
    <env:Body>
        <m:alert xmlns:m="http://example.org/alert">
            <m:msg>Pick up Mary at school at 2pm</m:msg>
        </m:alert>
    </env:Body>
</env:Envelope>
```

(http://www.w3.org/TR/2007/REC-soap12-part1-20070427/)
SOAP

SOAP nodes

- SOAP provides a distributed processing model using SOAP nodes
- A node may
  - transmit a SOAP message
  - receive a SOAP message
  - process a SOAP message
  - relay a SOAP message
- Use cases for SOAP nodes:
  - Crossing trust domains
  - Ensuring scalability
  - Providing value-added services along the message path

Example for a SOAP message path:

- Initial SOAP sender
- SOAP intermediary
- SOAP message
- SOAP intermediary
- SOAP message
- SOAP intermediary
- SOAP message
- SOAP receiver
The SOAP communication model supports four different styles:

- **2 communication styles**
  - remote procedure calls (RPC)
    - contains a procedure call
  - document exchange
    - contains only the actual message

- **2 encoding styles**
  - encoded
    - data type encoded in message
  - literal
    - refers to XML Schema

Four potential styles:
- RPC/Literal
- Document/Literal
- RPC/Encoded
- Document/Encoded

Only two are relevant according to the WS-I Basic Profile 1.0:
- document/literal
- RPC/literal

SOAP

SOAP body

- Carries the actual message payload
- Challenge to be solved in service oriented architectures
  - Which format to chose for a specific payload?
Excursus: RPC-style Web Services

- An RPC-style Web Service appears as a **remote object** to the client application.
- Clients express their request as a **method call** with a set of parameters.
- Messages are **automatically serialized/deserialized** back to the respective objects.
- Thus, RPC services are rather tightly coupled.
Example of RPC-style SOAP message

```xml
<?xml version="1.0" encoding="UTF-8"?>
<env:Envelope
    xmlns:env="http://www.w3.org/2003/05/soap-envelope"
    xmlns:m="http://www.supply.com/prices">
  <env:Header>
    <tx:Transaction-id
        xmlns:t="http://www.transaction.com/id"
        env:mustUnderstand="true">
      512
    </tx:Transaction-id>
  </env:Header>
  <env:Body>
    <m:GetProductPrice>
      <product-id>4893248</product-id>
    </m:GetProductPrice>
  </env:Body>
</env:Envelope>
```

The method name is hard wired.
• Document-style Web Services are **message driven**
• The client invokes the service by sending a **document** (e.g., Quote Request) rather than a discrete set of parameters (cf. RPC-style)
• The Web Service may (= synchronous) or may not (= asynchronous) return a response message (e.g., Quote)
• In an asynchronous case the client can continue computation without waiting for an immediate response
Example of document-style SOAP message

```xml
<?xml version="1.0" encoding="UTF-8"?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
  <env:Header>
    <tx:Transaction-id xmlns:t="http://www.transaction.com/id" env:mustUnderstand="true">
      512
    </tx:Transaction-id>
  </env:Header>
  <env:Body>
    <p:PurchaseOrder orderDate="2010-05-06" xmlns:p="http://www.supply.com/order">
      <p:from>
        <p:accountNumber>239dsd</p:accountNumber>
      </p:from>
      <p:to>
        <p:accountNumber>23540234</p:accountNumber>
      </p:to>
      <p:products> ... </p:products>
    </p:PurchaseOrder>
  </env:Body>
</env:Envelope>
```
SOAP

Error handling

- SOAP uses the `env:Fault` element to communicate faults to the originator of the faulty message or another SOAP node
- An `env:Fault` element has two mandatory sub-elements:
  - `env:Code`
  - `env:Reason`
  - `env:Detail` (optional, for application specific information)

- In case an error occurred, a SOAP message with an `env:Fault` element in the `env:body` is returned to the originator
SOAP

SOAP & HTTP

Client

Service listener

Service proxy

Application

Send HTTP POST with SOAP message

Receive request

Decode request message

Receive function call
  Do something
  Return value

Send response

Receive HTTP response with SOAP response
SOAP

Summary: Advantages

- **Simplicity**
  - Based on XML, highly structured, easy to parse
- **Portability**
  - No dependency on the underlying platform
- **Firewall-friendly due to HTTP**
- **Use of open standards**
- **Interoperability**
  - Because SOAP is based on XML and HTTP it is possibly the most widely interoperable protocol to date
- **Universal acceptance**
- **Resilience to changes**
  - Even if new versions of the specification are introduced, SOAP nodes may provide backward compatibility
SOAP

Summary: Disadvantages

- SOAP + XML over HTTP does not have a high-performance
  - However, this is a trade-off compared to its interoperability features
- SOAP is stateless (as is HTTP)
  - A requesting application must reintroduce itself to other applications when more connections are required
- SOAP serializes by value and does not support serialization by reference
  - Multiple copies of an object will occur over time, containing state information that is not synchronized with other dislocated copies of the same object
• Start Java program “ArithmeticService”

• Access it under http://localhost:8080/arithmeticservice

• Use SOAP-UI (http://www.soapui.org/) to access the service
WSDL

What is WSDL?

- Web Service Description Language

- XML-based specification schema for describing the public interface of a Web Service

- WSDL serves as a *contract* between a service provider and a client, invoking the service

- WSDL describes
  
  - *what* a service does; i.e., the operations the service provides
  - *where* it resides; i.e., details of the protocol specific address (URL)
  - *how* to invoke it, i.e., details of the data formats and protocols necessary to access the service's operations
WSDL

The two main parts of WSDL

- **Service interface definition (abstract part)**
  - describes the general Web service interface structure
  - the operations supported by the service
  - the operation parameters
  - and abstract data types

- **Service implementation part (concrete part)**
  - binds the interface
    - to a concrete network address
    - to a specific protocol
    - and to concrete data structures
  - a Web service client may bind to such an implementation and invoke the service
WSDL

WSDL specification

<definitions>

<types>
  data type definitions
</types>

<message>
  definition of the data being communicated
</message>

<portType>
  <operation>
    set of operations
  </operation>
</portType>

<binding>
  protocol and data format specification
</binding>

<service>
  location of the service
</service>

</definitions>
WSDL

WSDL Elements I

- `<definitions>`: WSDL root element
- `<types>`: Container for data type definitions used by the Web Service
  - Option A: Define a local XML Schema for parameter data types
  - Option B: Reference one or more existing external XML Schemas
  - Option C: Combination of Option A and Option B
- `<message>`: Definition of the messages, used by the Web Service. The type of a message is defined in `<types>`
- `<portType>`: Logical grouping of abstract `<operations>` which are supported by one or more endpoints
- `<operation>`: Operation signatures and I/O messages, defined in `<message>`. Consists of `<input>`, `<output>` and `<fault>` messages
WSDL

WSDL Elements II

- `<binding>`: Define the message format and the protocol details for each port type
  - `<soap:binding>`: Defines the SOAP style (RPC or document) and the SOAP protocol to use (usually HTTP)
  - `<operation>`: Defines each of the concrete operations, which the referenced port type exposes. For each operation the corresponding SOAP action is defined. Furthermore, it is defined how the input and output is encoded (literal or encoded). Default is literal.

- `<service>`: Defines the `<port>`s supported by the Web Service.
  - `<port>`: Refers to an existing `<binding>` and via the `<binding>` to a `<portType>`
  - `<soap:address>`: Defines the location, where the service may be accessed. (i.e., the URL)
WSDL

WSDL as part of a requestor-service interaction

Each message part is defined by some type, either custom defined or XSD provided.

Type-a
- custom defined
- int
- XSD built-in

A port exposes a service using a specific binding.

Input and output messages form an operation. A set of operations forms a port type.

A binding specifies how the operations are invoked using a specific protocol, e.g., SOAP.

A service is a collection of related endpoints (ports) that the client wishes to invoke.

Message
- part
- part

Client #A

Client #B

Java Implementation
Assume we want to provide the following functions via a Web Service

```java
public class MyWebService {
    public String greet(String name) {
        return "Hello " + name + "!";
    }
    public int addInt(int n1, int n2) {
        return n1 + n2;
    }
}
```

Then we obtain the following WSDL-document:
WSDL
Example (2/5)

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<definitions targetNamespace="http://endpoint.myws/" name="MyWebServiceService"
  schemas.xmlsoap.org/wsdl/soap/">
  <types/>

  <message name="addIntRequest">
    <part name="n1" type="xsd:int"/>
    <part name="n2" type="xsd:int"/>
  </message>

  <message name="addIntResponse">
    <part name="sum" type="xsd:int"/>
  </message>

  <message name="greetRequest">
    <part name="arg0" type="xsd:string"/>
  </message>

  <message name="greetResponse">
    <part name="return" type="xsd:string"/>
  </message>
</definitions>
```
<portType name="MyWebService">
  <operation name="addInt" parameterOrder="n1 n2">
    <input message="tns:addIntRequest"/>
    <output message="tns:addIntResponse"/>
  </operation>
  <operation name="greet" parameterOrder="arg0">
    <input message="tns:greetRequest"/>
    <output message="tns:greetResponse"/>
  </operation>
</portType>
<binding name="MyWebServicePortBinding" type="tns:MyWebService">
  <soap:binding
    transport="http://schemas.xmlsoap.org/soap/http" style="rpc"/>
  <operation name="addInt">
    <soap:operation soapAction="http://localhost:8080/addInteger"/>
    <input>
      <soap:body use="literal"
        namespace="http://endpoint.myws"/>
    </input>
  
    <output>
      <soap:body use="literal"
        namespace="http://endpoint.myws"/>
    </output>
  </operation>
  <operation name="greet">
    ...
  </operation>
</binding>

Arbitrary name
Points to the port type for the binding
May be used to identify a certain SOAP Request
WSDL
Example (4/5)

```xml
<binding name="MyWebServicePortBinding"
   type="tns:MyWebService">
   <soap:binding
       transport="http://schemas.xmlsoap.org/soap/http" style="rpc"/>
</binding>
```

**Document:**
- content of `<soap:Body>` is specified by XML Schema defined in the `<wsdl:type>` section.
- Does not need to follow specific SOAP conventions - the message is sent as one "document" in the `<soap:Body>` element.
- No additional formatting rules have to be considered. Document style is the default choice.

**RPC:**
- The structure of an RPC style `<soap:Body>` element needs to comply with the rules specified in detail in Section 7 of the SOAP 1.1 specification.
- According to these rules, `<soap:Body>` may contain only one element that is named after the operation, and all parameters must be represented as sub-elements of this wrapper element.
WSDL

Example (5/5)

```xml
<definitions>
  <service name="MyWebServiceService">
    <port name="MyWebServicePort">
      binding="tns:MyWebServicePortBinding">
        <soap:address location="http://localhost:8080"/>
      </soap:address>
    </port>
  </service>
</definitions>
```
1: One-way messaging
Sender
SOAP request message
Receiver

2: Request/response messaging
Sender
SOAP request message
Receiver
SOAP response message

3: Notification messaging
Sender
SOAP request message
Receiver

4: Solicit/response messaging
Sender
SOAP request message
Receiver
SOAP response message
One-way messaging

- The service endpoint receives a message, but does not send a response
- The `<operation>` element is declared with a single `<input>` element, but no `<output>` element
- Like a “fire and forget” mechanism
- A one-way-operation is typically thought of as asynchronous messaging

```xml
<!-- portType element describing the abstract interface of a Web Service -->
<wSDL:portType name="SubmitPurchaseOrderPortType">
  <wSDL:operation name="SubmitPurchaseOrder">
    <wSDL:input name="order" message="tns:SubmitPurchaseOrderMessage"
  
  </wSDL:input>
</wSDL:operation>
</wSDL:portType>
```
WSDL

2: Request/Response messaging

- The service endpoint receives a message and returns a message in response

- If an `<operation>` element is declared with a single `<input>` element followed by a single `<output>` element, it defines a request/response operation

```xml
<!– portType element describing the abstract interface of a Web Service -->
<wsdl:portType name="SubmitPurchaseOrderPortType">
  <wsdl:operation name="SubmitPurchaseOrder">
    <wsdl:input name="order" message="tns:SubmitPurchaseOrderMessage"/>
    <wsdl:output name="orderResponse" message="tns:PurchaseOrderResponseMsg"/>
  </wsdl:operation>
</wsdl:portType>
```
An operation in which the service endpoint sends a message to the client, but does not expect to receive a response is called a notification operation.

- This type of operation is used if services need to notify clients of events.
- A Web Service using the notification messaging pattern follows the push model of distributed computing (the client has registered with the Web service to receive messages about an event).
- Is the opposite of the one-way messaging pattern.
- The <operation> element contains an <output> tag, but no <input> message definitions.

```xml
<!– portType element describing the abstract interface of a Web Service -->
<wsdl:portType name="FloodWarningPortType">
  <wsdl:operation name="FloodWarning">
    <wsdl:output message="tns:FloodWarningMsg"/>
  </wsdl:operation>
</wsdl:portType>
```
**WSDL**

4: Solicit/response messaging

- An operation in which the service endpoint sends a message and expects to receive an answer-message in response
- Is the opposite of the request/response messaging pattern
- Is similar to notification messaging, except that the client is expected to respond to the Web Service
- As with notification messaging, clients of the solicit/response Web services must subscribe to the service in order to receive messages
- In the `<operation>` element first an `<output>` tag and then an `<input>` tag is declared

```xml
<!-- portType element describing the abstract interface of a Web Service -->
<wsdl:portType name="AnythingNewInYourBusinessPortType">
  <wsdl:operation name="AnythingNew">
    <wsdl:output message="tns:AnyThingNewRequest"/>
    <wsdl:input message="tns:MyNewsResponse"/>
  </wsdl:operation>
</wsdl:portType>
```
WSDL

Demo „Arithmetic Service“
UDDI

What is UDDI?

- Universal Description Discovery and Integration
- UDDI defines a scheme for publishing and finding Web services of business entities
- “Yellow Pages for Web Services”
- Thought as standard for one XML-directory for Web services

Three main pillars of a UDDI registry

- “white pages”: address, contact, and known identifiers
- “yellow pages”: industrial classification
- “green pages”: meta information on services (reference to service description in WSDL)

In 2005 IBM, Microsoft, and SAP closed their existing UDDI registries
UDDI
Reasons why UDDI never gained momentum

- Discovery model of UDDI is very limited
  - No complex queries on potential services possible
  - Annotation mechanism for service definitions is limited
- Pure technical focus (service definitions)
  - No integration of product data offered by a company
  - Missing B2B reference processes
  - Lack of quality and validity of entries
- Approach too generic (world-wide)
- Too many “fun-entries”

Although the public success of UDDI is very limited, UDDI may still leverage benefits in an intra-organizational context.
Java API for XML Web Services (JAX-WS)

Overview

- Java Specification Request 224: Java API for XML-Based Web Services (JAX-WS) 2.0: https://www.jcp.org/en/jsr/detail?id=224
- Web Services with JAX-WS 2 are POJOs, enriched with annotations:
  - @WebService
  - @SOAPBinding
  - @WebMethod
  - @WebParam
  - @WebResult
  - @OneWay
- XML processing using JAX-B (Java Architecture for XML Binding)
  - Marshalling: Java Object → XML instance
  - Unmarshalling: XML Instance → Java Object
- Java provides a simple mini server
Java API for XML Web Services (JAX-WS)

Application development approaches

- The WSDL to Java approach (contract first)
  - Defines the WSDL first
  - Use of tools such as `wsimport` to generate portable web service artifacts
  - Should be used for complex Web Service projects where multiple stakeholders and partners are involved.

- The Java to WSDL approach (contract last)
  - Create a Service Endpoint Interface (SEI) as Java source files
  - Use the source files as inputs to generate the WSDL and other required portable artifacts (using `wsgen`)
  - May be used for smaller Web Service projects with a limited set of requirements
Java API for XML Web Services (JAX-WS)

wsimport 101

- Use the WSDL of an existing Service to generate a Java Web Service Client

```
wsimport -d path/to/binaries -keep -s path/to/source -p my.java.package http://www.urltoservice.com?wsdl
```

d  path to the .class files
keep  flag indicating that the source files shall be kept
s  path for the source files
p  Java package for the generated classes
import javax.jws.WebMethod;
import javax.jws.WebResult;
import javax.jws.WebService;
import javax.jws.soap.SOAPBinding;

@WebService(name="ArithmeticService")
@SOAPBinding(style = SOAPBinding.Style.RPC)
public class ArithmeticService {

@WebMethod(operationName="addFunction")
@WebResult(name = "sum")
public int add(int addend_a, int addend_b) {
    return addend_a + addend_b;
}

@WebMethod(operationName="subtractFunction")
@WebResult(name = "difference")
public int subtract(int minuend, int subtrahend) {
    return minuend - subtrahend;
}
}
import javax.xml.ws.Endpoint;

public class RunService {

    public static void main (String [] args) {

        Endpoint endpoint = Endpoint.publish("http://localhost:8080/arithmeticservice", new ArithmeticService());
    }
}

Analyze the generated WSDL by opening
http://localhost:8080/arithmeticservice?wsdl
Further WS-Standards

- WS-Notification
- WS-Transfer
- WS-PolicyAssertions
- WS-Resource Framework
- WS-Security
- WS-SecureConversation
- WS-Policy
- WS-Trust
- WS-Federation
- WS-Privacy
- WS-Test
- WS-Eventing
- WS-MakeConnection
- ...

69
WS-* standard wrap up

- WS-death star

- Lots of different WS-* specifications have been developed

- Hard to cope with these standards, even with a software stack at hand

- While useful on an enterprise level, the WS-* approach is an over-engineering for smaller, more lightweight projects

- Alternative approaches using HTTP with plain XML or JSON are on the rise
RESTful Web Services

Introduction

• Acronym for REpresentational State Transfer
• Based on the dissertation of Dr. Fielding http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm
• Defines a set of architectural principles
• Centered around the concept of **resources** and the manipulation of a resource’s **state**

**Basic design principles**
- REST server provides access to resources
- REST client manipulates the resources using HTTP methods
- Statelessness (I did not say stateless applications!)
- Directory-like structure of URIs for addressing **resources**
- Transfer of XML or JSON data in order to alter the **state** of resources
RESTful Web Services

Use of HTTP methods

- Through HTTP methods resources may be accessed, modified, created, deleted
  - GET (retrieve a resource)
  - PUT (change the state of a resource, i.e., update a resource)
  - POST (create a new resource)
  - DELETE (remove a resource)

- Example: *Rename user Alice to Bob*

```
PUT /users/Alice HTTP/1.1
Host: myserver
Content-Type: application/xml
<?xml version="1.0"?>
<user>
  <name>Bob</name>
</user>
```
RESTful Web Services
Stateless 1/2

- Consider the following **stateful** design for accessing users in chunks of 10 record sets

```java
// Server-side code
currentChunk = getCurrentChunkForUser(432);
updateChunk(currentChunk, 432);
return currentChunk;
```

- Server must hold current state and client/server must use session mechanism in order to allow correct correlation to current state
  - Bad for clustered environments (session data synchronization across cluster)
  - Session overhead can become an issue (memory consumption)
  - `java.io.NotSerializableException` issues (in case of bad session design)
Consider the following **stateless** design for accessing users in chunks of 10 record sets:

```xml
<response chunk="4" nextChunk=""/>
  <user id="1">User 1</user>
  <user id="2">User 2</user>
  ....
</response>
```

- Responsibility for state management is entirely on the client's side.
- The server's response must allow the client to maintain the state, i.e., the response must contain information necessary for state management (see above).

See also: [http://www.peej.co.uk/articles/no-sessions.html](http://www.peej.co.uk/articles/no-sessions.html)
RESTful Web Services

Directory-like structure of URIs

- URIs are used to **address resources** (in order to change their state)
- In the context of RESTful Services URIs might be considered as self-documenting interfaces

- A resulting URI structure might be
  - [http://www.mycompany.com/users](http://www.mycompany.com/users)
  - [http://www.mycompany.com/users/internal](http://www.mycompany.com/users/internal)
  - [http://www.mycompany.com/users/internal/34](http://www.mycompany.com/users/internal/34)
  - [http://www.mycompany.com/users/external](http://www.mycompany.com/users/external)
RESTful Web Services
Transfer of XML or JSON data

- Resource representation reflects the current state of a resource
  - What values are currently stored for an entry $x$ at the time the client accesses the resource
  - E.g., 'request user xyz' returns the current state of the resource $user \ xyz$ at time $t$

- Client applications should be able to request the content in the format which best fits their needs
  - Use of MIME-Types in the HTTP Accept Header
  - e.g., application/json, application/xml, application/xhtml+xml
  - Also known as "content negotiation"

- JSON (JavaScript Object Notation)
  - Lightweight data interchange format
  - Easy for humans to read and write and easy for machines to parse and generate
  - Further information: [http://www.json.org/](http://www.json.org/)
RESTful Web Services

JSON example: person object

```json
{
    "name": "John Doe",
    "age": 43,
    "address": {
        "street": "Favoritenstraße 9-11",
        "city": "Vienna",
        "state": "AT"
    },
    "phoneNumbers": [
        {
            "type": "business",
            "number": "+43 58801 18800"
        },
        {
            "type": "mobile",
            "number": "+43 664 18800188"
        }
    ]
}
```
JAX-RS - Java API for RESTful Web Services
Building your own RESTful Services

- Sample implementation: Java Jersey: [https://jersey.java.net/](https://jersey.java.net/)
- Web Services with JAX-WS are POJOs, enriched with annotations:
  - @Path
  - @GET, @PUT, @POST, @DELETE,…
  - @Produces
  - @Consumes
  - …

```java
@Path("/users/{username}")
public class UserResource {

  @GET
  @Produces("text/xml")
  public String getUser(@PathParam("username") String userName) {
      ...
  }
}
```
JAX-RS - Java API for RESTful Web Services
HEAD and OPTION Request

- HTTP HEAD: invokes the implemented GET method (if present), but does not return the response entity
- HTTP OPTION:
  - sets the allow response header to the HTTP methods, supported by the resource
  - in addition the description of the RESTful Web Service using WADL (Web Application Description Language) is returned

```xml
<application xmlns="http://research.sun.com/wadl/2006/10">
  <doc xmlns:jersey="http://jersey.java.net/" jersey:generatedBy="Jersey: 1.8"/>
  <resources base="http://localhost:8080/we-restful-service/rest/">
    <resource path="students">
      <method id="getStudentsXML" name="GET">
        <response>
          <representation mediaType="application/xml"/>
        </response>
      </method>
      <method id="getStudentsJSON" name="GET">
        <response>
          <representation mediaType="application/json"/>
        </response>
      </method>
      ...  
    </resource>
  </resources>
</application>
```
REST

Demo

- Accessing Facebook's API
  - e.g., [https://graph.facebook.com/tuwien](https://graph.facebook.com/tuwien)
- Accessing sample student application
  - see [https://github.com/pliegl/we2014](https://github.com/pliegl/we2014) for further details

<table>
<thead>
<tr>
<th>Resource</th>
<th>POST create</th>
<th>GET read</th>
<th>PUT update</th>
<th>DELETE delete</th>
</tr>
</thead>
<tbody>
<tr>
<td>/students</td>
<td>Create a new student</td>
<td>List all students</td>
<td>Bulk update a set of students</td>
<td>Delete all students</td>
</tr>
<tr>
<td>/student/1</td>
<td>Error</td>
<td>Show student with register number 1</td>
<td>If exists update student with register number 1</td>
<td>Delete student with register number 1</td>
</tr>
</tbody>
</table>

[http://localhost:8080/we-restful-service/rest/students](http://localhost:8080/we-restful-service/rest/students)
Summary

- Web services support interoperable machine-to-machine interaction
- Web services may be part of Web applications, but do not have a GUI themselves
- Web services are used for integration purposes
- 3 core standards for Web services
  - SOAP
  - WSDL
  - UDDI
- RESTful services are another powerful architectural style for realizing machine-to-machine interaction
Interesting Literature & Tools

  - recommended - this presentation is mostly based on this book

- Martin Kalin, *Java Web Services*, O'Reilly, 2009
  - Java-specific perspective on Web Services

- Eben Hewitt, *Java SOA Cookbook*, O'Reilly, 2009
  - Java-specific perspective on SOA – covers additional concepts such as service orchestration, etc. as well

- www.soapui.org/
  - Useful for Web Service testing
Interesting Literature & Tools – RESTful Web Services

- Leonard Richardson and Sam Ruby, *RESTful Web Services*, O'Reilly, 2007

- Brian Mulloy, Web API Design – Crafting interfaces that developers love, apigee

- Cesare Pautasso and Erik Wilde, Tutorial Design Principles, Patterns and Emerging Technologies for RESTful Web Services
  - [http://dret.net/netdret/docs/rest-icwe2010/](http://dret.net/netdret/docs/rest-icwe2010/)

- Google Chrome Advanced REST client
  - [https://chrome.google.com/webstore/detail/advanced-rest-client/hgmloofddffdnphfgcellkdfbfbjeloo](https://chrome.google.com/webstore/detail/advanced-rest-client/hgmloofddffdnphfgcellkdfbfbjeloo)