

Web Services

Nicoleta Preda

January 11, 2017

Web Evolution

Technology	TCP/IP	HTML	XML
Purpose	Connectivity	Presentation	Programmability
Applications	E-Mail, FTP...	Web Pages	Web Services
Outcome	Create the Web	Browse the Web	Program the Web

Web Service (WS)

- ▶ A way of publishing/exporting data in the Internet
- ▶ A **Web Service** consists of several **functions** (methods, operations)
- ▶ Observation: We sometimes use Web Service to refer to a function

XML Overview

- ▶ XML is a language for building languages.
- ▶ Basic rules: be well formed and be valid
- ▶ Particular XML "dialects" are defined by an XML Schema.
- ▶ XML itself is defined by its own schema
- ▶ XML is extensible via namespaces

Advantage: Many basic tools available:

- ▶ parsers: SAX, DOM
- ▶ query languages: XPath and XQuery
- ▶ transformation languages: XSLT

XML & Web services

XML provides a natural substrate for distributed computing

- ▶ Its just a data description
- ▶ Programming language independent of the platform

Plan

SOAP Services

REST Services

Chapter Plan

- ▶ Web Services Architecture
- ▶ SOAP (messaging)
- ▶ WSDL (service description)
- ▶ UDDI (registry)

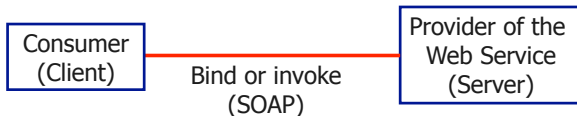
Web Services: Basic Architecture

The simplest Web service system has two participants:

A service **producer** (provider) - server

A service **consumer** (requester) - client

The provider presents the interface and implementation of the service, and the requester uses the Web service.



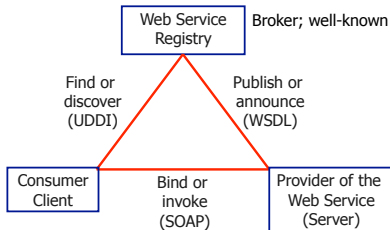
Web Services Architecture

A more sophisticated system:

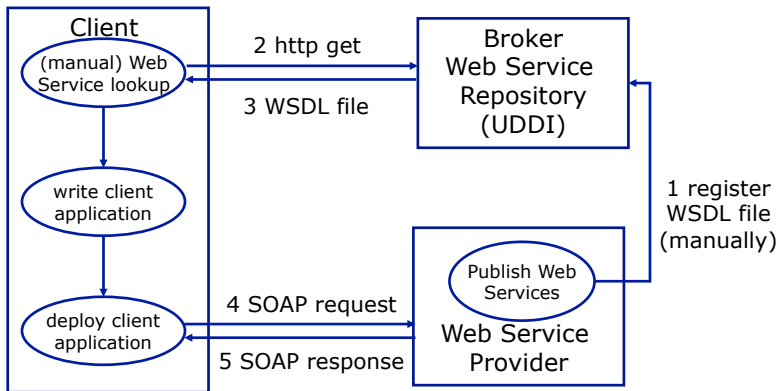
A **registry** - acts as a broker for Web Services

A provider - can publish services to the registry

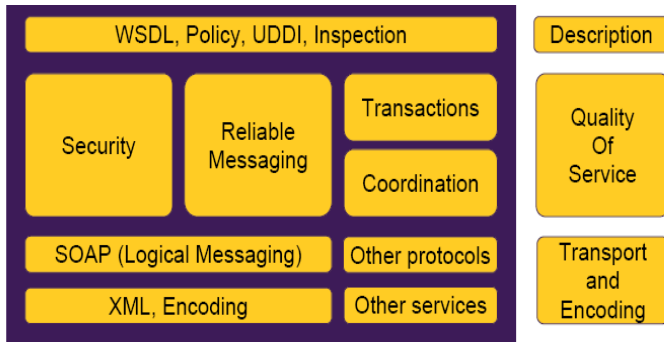
A consumer - can then discover services in the registry



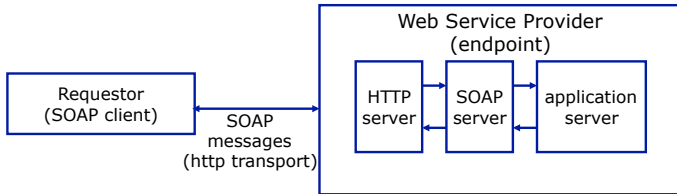
Basic Web Service Usage Scenario



Stack of Web Service Standards



Web Services Implementation

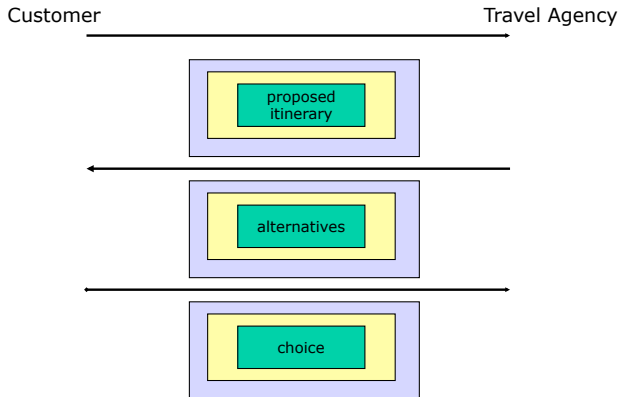


- ▶ Application Server (Web Service-enabled)
provides implementation of services and exposes it through WSDL/SOAP implementation in Java, as EJB, as .NET (C#) etc.
- ▶ SOAP server
implements the SOAP protocol
- ▶ HTTP server
standard Web server
- ▶ SOAP client
implements the SOAP protocol on the client site

Chapter Plan

- ▶ Web Services Architecture
- ▶ SOAP (messaging)
- ▶ WSDL (service description)
- ▶ UDDI (registry)

Conversational Message Exchanges in SOAP



SOAP: Simple Object Access Protocol

- ▶ Lightweight messaging framework based on XML
- ▶ Supports simple messaging and RPC
- ▶ SOAP consists of:
 - Envelope construct: defines the overall structure of messages
 - Encoding rules: define the serialization of application data types
 - SOAP RPC: defines representation of remote procedure calls and responses
 - Binding framework: binding to protocols such as HTTP, SMTP
 - Fault handling
- ▶ Soap supports advanced message processing:
 - forwarding intermediaries: route messages based on the semantics of message
 - active intermediaries: do additional processing before forwarding messages, may modify message

SOAP messages

- ▶ SOAP messages consist of

 - Envelope: top element of XML message (required)

 - Header: general information on message such as security (optional)

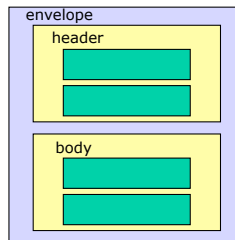
 - Body: data exchanged (required)

- ▶ Header

 - elements are application-specific
 - may be processed and changed
by intermediaries or recipient

- ▶ Body

 - elements are application-specific
 - processed by recipient only



Skeleton SOAP Message

```
<?xml version="1.0"?>
<soap:Envelope
  xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
  soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
  <soap:Header>
    ...
  </soap:Header>
  <soap:Body>
    ...
    <soap:Fault>
      ...
    </soap:Fault>
  </soap:Body>
</soap:Envelope>
```

Skeleton SOAP Message

```
<env:Envelope xmlns:ens="http://www.w3.org/2002/12/soap-envelope">
```

```
<env:Header>
```

```
<m:reservation xmlns:m=http://travelcompany.org/reservation
  env:role=http://www.w3.org/2002/12/soap-envelope/role/next env:mustUnderstand="true">
  <m:dateAndTime>2015-11-29T14:20:00.000-05:00</m:dateAndTime>
</m:reservation>
```

```
<n:passanger xmlns:n=http://travelcompany.com/employees
  env:role="http://www.w3.org/2002/12/soap-envelope/role/next" env:mustUnderstand="true">
<n:name> Alice Ruberg </n:name>
```

```
</env:Header>
```

```
<env:Body>
```

```
<p:itinerary xmlns:p="http://travelcompany.example.org/reservation/travel">
  <p:departure>
    <p:departing>New York</p:departing>
    <p:arriving>Los Angeles</p:arriving>
    <p:departureDate>2015-12-19</p:departureDate>
  </p:departure>
</p:itinerary>
```

```
</env:Body>
```

SOAP RPC: Remote Procedure Call

Encapsulate RPC into SOAP messages

- ▶ procedure name and arguments
- ▶ response (return value)
- ▶ processing instructions (transactional RPC!)

```
<?xml version='1.0' ?>
<env:Envelope xmlns:env="http://www.w3.org/2002/12/soap envelope">
  <env:Header>
    <t:transaction xmlns:t="http://thirdparty.example.org.transaction"
      env:encodingStyle="http://example.com/encoding" env:mustUnderstand="true">
      5</t:transaction>
  </env:Header>
  <env:Body>
```

```
    <m:chargeReservation env:encodingStyle="http://www.w3.org/2002/12/soap-encoding"
      xmlns:m="http://travelcompany.org/">
      <m:reservation xmlns:m="http://travelcompany.org/reservation"> <m:code>DDXZF</m:code>
      </m:reservation>
      <o:creditCard xmlns:o="http://travelcompany.com/financia">
        <n:name xmlns:n="http://travelcompany.com/employee"> Alice Ruberg </n:name>
        <o:number>1473484265576</o:number> <o:expiration>2016-02</o:expiration>
      </o:creditCard>
    </m:chargeReservation>
```

```
</env:Body>
</env:Envelope>
```

SOAP RPC: Remote Procedure Call

Encapsulate RPC into SOAP messages

- ▶ procedure name and arguments
- ▶ response (return value)
- ▶ processing instructions (transactional RPC!)

```
<?xml version='1.0' ?>
<env:Envelope xmlns:env="http://www.w3.org/2002/12/soap envelope">
  <env:Header>
    <t:transaction xmlns:t="http://thirdparty.example.org.transaction"
      env:encodingStyle="http://example.com/encoding" env:mustUnderstand="true">
      5</t:transaction>
    </env:Header>
    <env:Body>
```

```
      chargeReservation(code:DDXZF,
                        creditCard{name: Alice Ruberg,
                                   number: 1473484265576,
                                   expiration: 2016-02}
                        )
```

```
</env:Body>
</env:Envelope>
```

SOAP Response

```
<?xml version='1.0' ?>
<env:Envelope xmlns:env="http://www.w3.org/2002/12/soap-envelope" >
<env:Header>
  <t:transaction xmlns:t="http://thirdparty.example.org/transaction
    env:encodingStyle="http://example.com/encoding
    env:mustUnderstand="true">5</t:transaction>
</env:Header>
<env:Body>
```

```
<m:chargeReservationResponse
  env:encodingStyle="http://www.w3.org/2002/12/soap-encoding
  xmlns:m="http://travelcompany.example.org/">
  <m:code>FT35ZBQ</m:code>
  <m:viewAt> http://travelcompany.example.org/reservations?code=FT35ZBQ
</m:viewAt>
</m:chargeReservationResponse>
```

```
</env:Body>
</env:Envelope>
```

The *fault* element

Carries an error message

If present, must appear as a child of <Body>

Must only appear once

Has the following sub-elements:

Sub Element	Description
<faultcode>	A code for identifying the fault (VersionMismatch, MustUnderstand, Client, Server)
<faultstring>	A human readable explanation of the fault
<faultactor>	Information about who caused the fault to happen
<detail>	Holds application specific error information related to the Body element

Protocol Binding

- ▶ Bindings to different protocols possible: HTTP, SMTP
- ▶ Different HTTP bindings:
 - HTTP POST (for request-response communications)

HTTP GET

Request

```
POST /Reservations?code=FT35ZBQ HTTP/1.1
Host: travelcompany.example.org
Content-Type: application/soap+xml; charset="utf-8"
Content-Length: nnnn
<?xml version='1.0' ?>
<env:Envelope xmlns:env="http://www.w3.org/2002/12/soap-envelope">
...SOAP request message...
</env:Envelope>
```

Response

```
HTTP/1.1 200 OK
Content-Type: application/soap+xml; charset="utf-8"
Content-Length: nnnn
<?xml version='1.0' ?>
<env:Envelope xmlns:env="http://www.w3.org/2002/12/soap-envelope">
... SOAP response message ...
</env:Envelope>
```

Chapter Plan

- ▶ Web Services Architecture
- ▶ SOAP (messaging)
- ▶ WSDL (service description)
- ▶ UDDI (registry)

WSDL: Web Service Definition Language

Define a Web Service in WSDL by

Writing an XML document conforming to the WSDL specs

Describes three fundamental properties:

- ▶ What are the **operations (functions, methods)** provided by the service
- ▶ How a service is **accessed**
Data format and protocol details
- ▶ Where a service is **located**
Address (URL) details

Overview of Defining WSDL Services

1. Define in XML Schema the message types used when invoking the service:

MT1, MT2 etc.

2. Define (named) messages by using these types, e.g.

message m1 has type MT1, message m2 has type MT2 etc.

3. Define Services that consist of one or more operations; each operation is implemented by the exchange of messages

service S offers operation O1;

for executing O1 first send a request message m1,

then a response message m2 is returned

4. Define a Binding B to a specific protocol, e.g.

SOAP service S is implemented in SOAP;

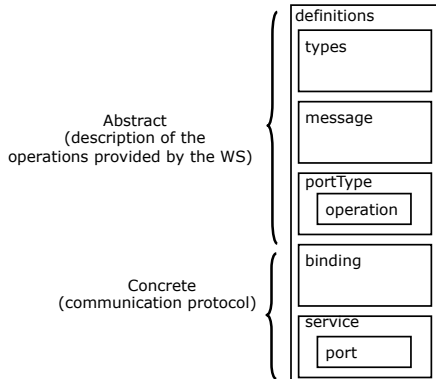
the SOAP messages are constructed from the abstract messages m1 and m2 by,

e.g. inlining the messages as body of SOAP message

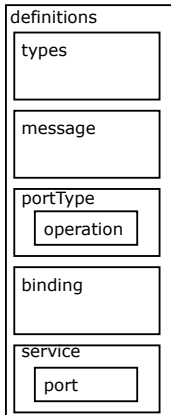
5. Service S is provided with binding B at the following URI's (called ports)

WSDL – Web Service Description Language

- ▶ Description of Web services in XML format
 - ▶ abstract description of operations and their parameters (messages)
 - ▶ binding to a concrete network protocol (e.g. SOAP)
 - ▶ specification of endpoints for accessing the service
- ▶ Structure of a WSDL document:



WSDL Document Structure



All the data types used by the Web service

Parameters and messages used by method

Abstract interface definition: each operation element defines a method signature

Binds abstract methods to specific protocols

A service is a collection of ports.

A port is a specific method and its URI

Overall XML Document Structure

```
<definitions name="TravelAgency">
  <types>
    <schema> definition of types in XML Schema ... </schema>
  </types>
  <message name="ChargeReservation"> XML-schema definition </message>
  <message name="ChargeReservationResponse"> XML-schema definition </message>
  <portType name="TravelAgencyPortType">
    <operation name="ChargeReservation"> definition of an operation ... </operation>
  </portType>
  <binding name="TravelAgencySoapBinding"> definition of a binding ... </binding>
  <service name="TravelAgencyService">
    <port name="TravelAgencyPort"> definition of a port ... </port>
  </service>
</definitions>
```

XML Schema

- ▶ Grammar (data definition language) for specifying valid documents
- ▶ Uses same syntax as regular XML documents: verbose and difficult to read
- ▶ Provides local scoping of subelement names
- ▶ Incorporates namespaces
- ▶ Types
 - ▶ primitive types: string, integer, float, date, ...
 - ▶ simpleType constructors: list, union
 - ▶ restrictions: intervals, lengths, enumerations, regex patterns,
- ▶ Flexible ordering of elements
- ▶ Key and referential integrity constraints

Example Type Element

```
<types>
<xsd:schema targetNamespace="http://travelcompany.com/ns"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:complexType name="creditCard">
    <xsd:sequence>
      <xsd:element name="name" type="xsd:string">
      <xsd:element name="number" type="xsd:integer">
      <xsd:element name="expiration" type="xsd:date">
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
</types>
```

WSDL to Code

- ▶ Convert WSDL document to code
E.g., Apache AXIS WSDL2java
- ▶ Derive WSDL from Java classes
E.g., Apache WSDL, Eclipse plug-in

Chapter Plan

- ▶ Web Services Architecture
- ▶ SOAP (messaging)
- ▶ WSDL (service description)
- ▶ UDDI (registry)

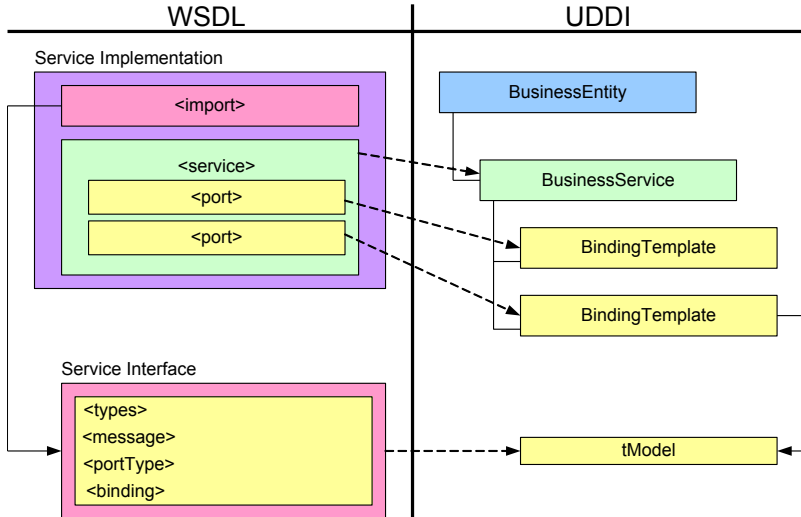
UDDI: Initial Goals and Final Use

- ▶ Goal: provide access to WSDL descriptions registered to a broker
- ▶ UDDI is an open industry initiative (IBM, Microsoft, SAP)
- ▶ The work at UDDI was stopped in 2006 due to the lack of wide acceptance
- ▶ Nowadays, UDDI systems are most commonly found inside companies

UDDI: Universal Description Discovery and Integration

- ▶ Standard for describing, publishing and finding web services
 - Use XML-based description files for services
- ▶ Main components
 - White pages: basic contact information about an organization
 - Yellow pages: classification of organization based on industrial categorization
 - Green pages: technical description of services offered by registered organizations
- ▶ Access to UDDI Registry
 - Standard UDDI API (accessible via SOAP)
 - Web browser
- ▶ Data Structures (XML)
 - Business entity: general information + business services
 - Business services: business level description + binding templates
 - Binding templates: access point + tModel (service types)
 - tModel: abstract definition of a web service

WSDL – UDDI



Web Services within the Distributed Computing Environment

- ▶ Loosely Coupled

Each service exists independently of the other services that make up the application.

Individual pieces of the application to be modified without impacting unrelated areas.

- ▶ Ease of Interface

Data is isolated between applications creating "silos".

Web Services act as glue between these and enable easier communications within and across organisations.

Plan

SOAP Services

REST Services

REST: REpresentational State Transfer

- ▶ Introduced in Roy Fielding's doctoral dissertation (2000)
- ▶ Fielding contributed also to the design of HTTP and URI's

Real REST-based Web Services

ProgramableWeb.com:

- references more than 12000 APIs from various domains (in 2015)
- > 70% of the publicly available Web Services are REST

Search (3200 APIs)

Traveling (1200 APIs)

Social (3000 APIs)

Music (1000 APIs)

Financial (1200 APIs)

Science (600 APIs)

Weather (300 APIs)



Expedia Affiliate
Network



A Call to a Real Web Service

Web site: MusicBrainz

Web Service function: "get album by artist name"

Web call for input "Rihanna": <http://musicbrainz.com?artist=Rihanna>

Result for the Web call:

```
<release-group id="a7753ec7-2fea-3621-a73b-e5d3462d4741"  
  <primary-type>Album</primary-type>  
</release-group>  
<date>2002-09-16</date>  
<country>GB</country>  
<release-event-list>  
  <release-event>  
    <date>2002-09-16</date>  
    <area id="8a754a16-0027-3a29-b6d7-2b40ea0481ed">  
      <name>United Kingdom</name>  
      <sort-name>United Kingdom</sort-name>  
      <iso-3166-1-code-list>  
        <iso-3166-1-code>GB</iso-3166-1-code>  
      </iso-3166-1-code-list>  
    </area>
```

REST Architecture

- ▶ REST is not protocol specific
- ▶ However, it is usually associated with HTTP

HTTP

- ▶ Synchronous request/response network protocol
- ▶ Used for collaborative distributed document-based systems
- ▶ The documents are encoded using different formats:
 - XML
 - JSON
 - Binary data may be included in the documents bodies
- ▶ Browsers usually use only a small part of HTTP

REST Architectural Principles

- ▶ The web has **addressable** resources:
 - Each resource has a URL.
- ▶ The web has a **uniform interface**. E.g.,
 - HTTP has a small number of methods.
 - Use these to manipulate resources.
- ▶ The Web is representation oriented - providing diverse formats:
 - XML
 - JSON
- ▶ **Stateless communication**: using features of the Web methods
- ▶ **Hypermedia** is used as the engine of application state

Principle: Addressability

Addressability (not restricted to HTTP)

Each HTTP request uses a URI.

The format of a URI is well defined:

`scheme://host:port/path?queryString#fragment`

scheme	- not necessarily HTTP; may be FTP or HTTPS
host	- field may be a DNS name or a IP address
port	- may be derived from the scheme; HTTP implies port 80
path	- set of text segments delimited by the "/"
queryString	- list of parameters represented as name=value pairs - each pair is delimited by an "&" - space is represented with the '+' characters - for special characters use % followed by two hex digits
fragment	- used to point to a particular place in the document.

Principle: Uniform Interface

► HTTP GET

- read only operation
- idempotent (once same as many)
- safe (no important change to server's state)
- may include parameters in the URI

`http://www.travelagency.com/flights?source=Bucharest&dest=Paris`

► HTTP PUT

- store the message body
- insert or update
- idempotent
- not safe

Principle: Uniform Interface

► HTTP POST

- not idempotent
- each method call may modify the resource in a unique way
- the request may or may not contain additional information
- the parameters are found within the request body (not in the URI)

► HTTP DELETE

- remove the resource
- each method call may modify the resource in a unique way

Advantages of the Uniform Interface

- ▶ Familiarity
 - No need to use a complicated syntax to describe methods signatures.
 - We already know the methods.
- ▶ Interoperability
 - SOAP-WS has been a moving target.
 - HTTP is widely supported.
- ▶ Scalability
 - Since GET is idempotent and safe.
 - Results may be cached by clients or proxy servers.

Principle: Representation Oriented

- Representations of resources are exchanged
- Representations may be in many formats: XML, JSON, YAML, etc
- GET returns a representation
- PUT and POST passes representations to the server so that underlying resources may change

How the representation mode is specified?

- ▶ HTTP uses the CONTENT-TYPE header for the call results
- ▶ The value of the CONTENT-TYPE is a MIME typed string. Versioning information may be included

Principle: Communicate Statelessly

- ▶ The application may have a state but there is no client session data stored on the server.
- ▶ If there is any session-specific data it should be held and maintained by the client and transferred to the server with each request as needed.
- ▶ The server is easier to scale. No replication of session data concerns.

Hypermedia as the Engine of Application State

- ▶ Hypermedia consists of documents with **links**
- ▶ Hence, a REST Web service can use **links** in call results to encode
 - the interactions that can be preformed next
 - possible transitions of the current state of the application

```
<order id = "111">  
  <customer> http://travelagency.com/customers/customer/1234  
  </customer>  
  <cancel> http://travelagency.com/canceling?orderId=111  
  </cancel>  
  <booking>  
    <booking>  
      <persons>2  
      <train>  
        http://travelagency.com/trains/trains/8754  
      </train>
```

SOAP-based vs. REST-based Web services

- SOAP-WSs use HTTP strictly as a transport protocol
- REST-WSs used the application protocol to express calls to operations:

SOAP	
Request	Response
XML (SOAP)	XML (SOAP)

REST	
Request	Response
URI	XML

- SOAP-WSs: requests/responses are typed using XML-Schema
the error response is specified
- REST-WSs: the service does not communicate the schema
that it used to encode the responses