
Conception of Information Systems
Lecture 9: Overview of Web Services

10 May 2005

<http://lsirwww.epfl.ch/courses/cis/2005ss/>

Outline

1. The Big Picture
2. SOAP
3. WSDL
4. UDDI
5. *Overview of the JAX-RPC Web Service Architecture*
6. *Building and Deploying a JAX-RPC Web Service*
7. *Building and Running a JAX-RPC Client*

7.1 The Big Picture

- "Web Services are a new breed of Web application. They are self-contained, self-describing, modular applications that can be published, located, and invoked across the Web. Web Services perform functions, which can be anything from simple requests to complicated business processes. ... Once a Web Service is deployed, other applications (and other Web Services) can discover and invoke the deployed service." *IBM tutorial on Web Services*
- "Comparing CORBA with the ongoing work on Web Service technologies, we are forced to ask the question: are we reinventing the wheel ? A. Gokhale, B. Kumar, A. Sahuguet; *Reinventing the Wheel ? CORBA vs. Web Services*
- Distributed computing model based on asynchronous messaging (XML)
 - Support dynamic application integration over the Web
 - Web Services connect computers and devices with each other using the Internet to exchange data and access services
 - On-the-fly software creation through the use of loosely coupled, reusable software components
 - Software can be delivered and paid per-use as opposed to packaged products.
 - Business services can be distributed over the Internet

©2004-2005, Karl Aberer & J.P. Martin-Flatin

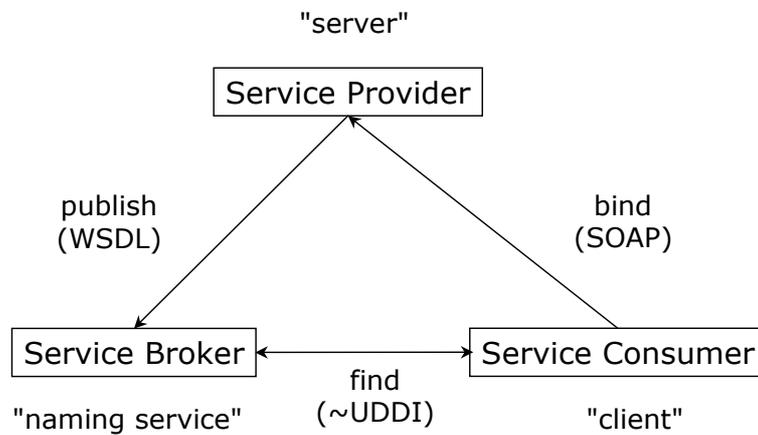
3

The notion of Web Services captures a recent technological development that aims at the possibility to access services (i.e. applications) through the Internet infrastructure (i.e. Web standards). Technologically they represent only a small change as compared to earlier distributed computing standards, but the impact of this new technology is considered as being very high as it facilitates automated distributed computing and information access by taking advantage of the ubiquitously available Web infrastructure. In particular on-demand use of software is considered as one of the important developments Web Service technology is going to support.

Web Services Design Principles

- Web-based Protocols (≠ CORBA)
 - Web-services based on HTTP
 - protocols can traverse firewalls, can work in a heterogeneous environment
- Interoperability
 - SOAP defines a common standard that allows different systems to interoperate
- XML-based (XML schema) (≠ CORBA)
 - machine-readable documents
- Modularity
 - Service Components are useful in themselves, reusable, composable
- Availability
 - Services are available to systems that wish to use them
 - Services must be exposed outside of the particular system they are available in
- Machine-readable description
 - used to identify the interface, the location and access information
- Implementation-independence
 - Service interface available independent of the ultimate implementation
- Published
 - Searchable service repositories of service descriptions

Here we summarize some of the design principles that underlie Web Service technology. Note that many of these principles also hold for other distributed computing technologies, such as CORBA, but that the use of ubiquitously available Web technology makes the infrastructure easier accessible and wider available.



A service-oriented architecture consists of three components:

- *service providers* that publish available services and offer bindings for services
- *service brokers* that allow service providers to publish their services (register and categorize). They provide also mechanisms to locate services and their providers
- a *service consumer* that uses the service broker to find a service and then invokes (or binds) the service offered by a service provider.

Note that this architecture maps 1:1 onto the CORBA model of server-naming service-client.

For each of the three activities the Web Services architecture provides specific standards, namely SOAP, WSDL, and (at least supposedly) UDDI. SOAP and WSDL are W3C standards, whereas UDDI is an OASIS standard.

Web Services Architecture



- A set of standards for implementing Web Services
 - Publication and Discovery: UDDI (Universal Description, Discovery and Integration))
 - Service Description: WSDL (Web Services Description Language)
 - Messaging and RPCs: SOAP (Simple Object Access Protocol)
 - Transfer Protocol: HTTP
- **UDDI** provides a mechanism for clients to find Web Services
 - a UDDI registry is similar to a CORBA trader
- **WSDL** defines services as collections of network endpoints or *ports*
 - a port is defined by associating a network address with a binding (servers)
 - a collection of ports define a service
- **SOAP** is a message layout specification that defines a uniform way of passing XML-encoded data and to bind to HTTP as the underlying communication protocol
 - SOAP is basically a technology to allow for "RPC *over the Web*"

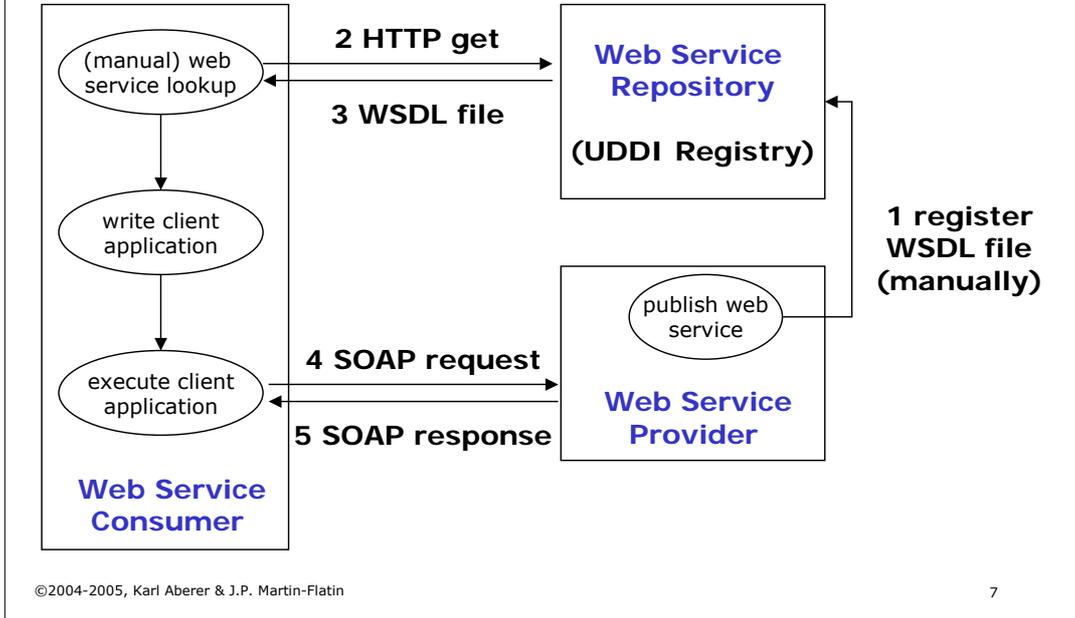
©2004-2005, Karl Aberer & J.P. Martin-Flatin

6

The different standards of the Web Services architecture establish a communication infrastructure that can be viewed as a high-level protocol stack, where each higher level protocol builds on the lower level protocol:

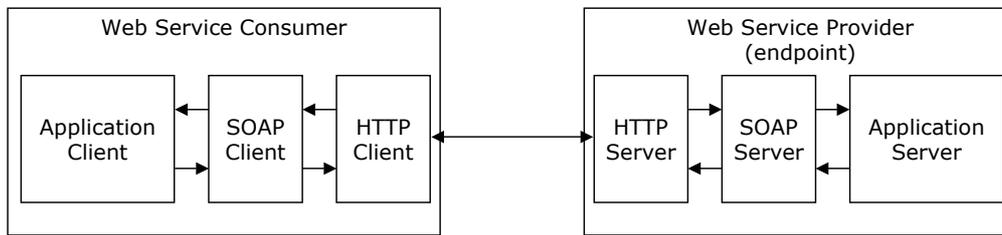
- The *Web Service transport layer* is responsible for the basic communication between Web Services and relies on existing application-layer protocols (usually HTTP).
- The *messaging layer* uses the XML-based SOAP protocol for exchanging messages in a request/reply fashion in a distributed environment.
- The *service description layer* allows describing interfaces of Web Services – including operations and their parameters. The descriptions are based on the XML-based WSDL language.
- The *publication and discovery layer* provides mechanisms for service brokers. This layer was initially based on UDDI, a standard that defines XML-based service AND business descriptions. UDDI did not live up to expectations and has encountered little success so far.

Simple Web Service Usage Scenario



This figure illustrates a (possible) basic Web Service usage scenario. A Web Service provider registers its Web Service at a UDDI repository. This could be done manually through a Web interface or through a UDDI API. He registers the description of the Web Service which is given in WSDL. A potential user may, for example, manually look up in the UDDI repository the Web Service through http and obtain the WSDL file. This file contains all the information needed in order to access the Web Service. Based on this information he/she implements a client application that makes use of the Web Service. When the client application is executed it accesses the Web Service by using the SOAP protocol for service invocation.

Web Services Implementation



- **Application Server (WS-enabled)**
 - provides implementation of services and exposes it through WSDL/SOAP
 - implementation in Java for J2EE, in C# for .NET, etc.
- **SOAP server**
 - implements the SOAP protocol
- **HTTP server**
 - standard Web server
- **SOAP client**
 - implements the SOAP protocol on the client site

©2004-2005, Karl Aberer & J.P. Martin-Flatin

8

The implementation of Web Services (at runtime) requires in particular the support of the SOAP protocol on the server and the client site. At the server this support is provided by a SOAP server that uses the HTTP server for transporting SOAP messages. The applications implementing the service can be implemented in any implementation platform, ranging from simple programming languages to application services, that support a Web Service front-end. It is irrelevant what exactly the interface between the SOAP server and application server is and to which degree exposing applications as Web Services is automated (e.g. whether Web Services can be automatically generated from application server components such as EJBs). These solutions are vendor-specific.

Comparison between CORBA and Web Services (1/2)



CORBA	Web Services
IDL	WSDL
CORBA Naming Service	~UDDI
CORBA Stubs/Skeletons	SOAP Messaging
CDR binary encoding	XML Unicode encoding
IIOP	HTTP
TCP/IP	TCP/IP

©2004-2005, Karl Aberer & J.P. Martin-Flatin

9

It is interesting to compare the Web Services stack with the CORBA stack. This clearly shows that the two technologies address exactly the same problem using two different approaches. Also observe that both technologies are based on TCP/IP thus are Internet-based, but they deviate already at the level of the transfer protocol: where Web Services use a standard Web protocol, CORBA is based on a specific protocol (IIOP).

Comparison between CORBA and Web Services (2/2)

Feature	CORBA	Web Services
Data Model	Object Model	SOAP message exchange model
Client-Server Coupling	Tight coupling	Loose coupling
Type system	IDL (static or dynamic)	XML
Location transparency	Object references	URL
Parameter passing	By reference/value	By value only
Type checking	Static+runtime	Runtime only
Service discovery	Naming/Trading service	~UDDI
Serialization	Built into ORB	Chosen by user

©2004-2005, Karl Aberer & J.P. Martin-Flatin

10

It is also interesting to compare in more detail how CORBA and Web Services address specific technical issues.

Data Model: Web Services model applications using a message-based paradigm, i.e. message exchanges among endpoints (service providers). CORBA is based on strong coupling between clients and servers, modeled as method invocations among objects.

Parameter passing: since Web Services have no notion of objects also no references can be passed. All data is passed by value.

Type Checking: CORBA supports both static (which is the standard) and dynamic type checking (supported by static stubs/skeletons resp. by using the DII)

State: Since CORBA uses a synchronous communication model during the interaction a common state is established. Web Services are a priori stateless. However, there exist extensions to implement stateful Web processes based on Web Services (more on that later in the part on workflows)

Service Discovery: The UDDI of Web Services provides a substantially more powerful approach to model and search Web Services, as compared to Naming services in CORBA.

Serialization: With Web Service the serialization of application data (in XML) can be chosen by the user.

7.2 SOAP – Simple Object Access Protocol

- Lightweight message-based communication framework based on XML
- Supports simple messaging and RPCs
- SOAP consists of
 - Envelope: defines the overall structure of the message
 - Encoding rules: define the serialization of application data types
 - SOAP RPC: defines representation of remote procedure calls and responses
 - Binding framework: binding to transfer protocols (e.g., HTTP)
 - 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

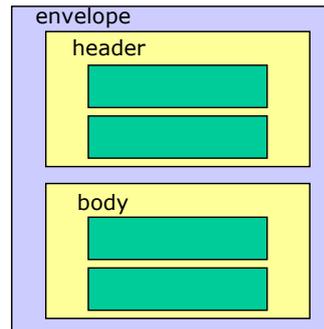
SOAP defines the protocol by which Web Services can be invoked. It supports both simple messaging (as in message queuing approaches) and RPC-style communication. It can be used over any transport protocol layer (such as HTTP, SMTP). SOAP defines the basic structure of messages and encoding rules for data types (used as parameters in procedure/method calls) and the encoding of procedure calls and responses. SOAP also defines bindings to specific transport protocols most notably to HTTP and SMTP.

SOAP is designed to also allow the implementation of advanced message processing models, in particular the use of intermediaries. Intermediaries both can just route messages, based on the content of the message, or do some additional processing before routing the message.

Structure of SOAP Messages

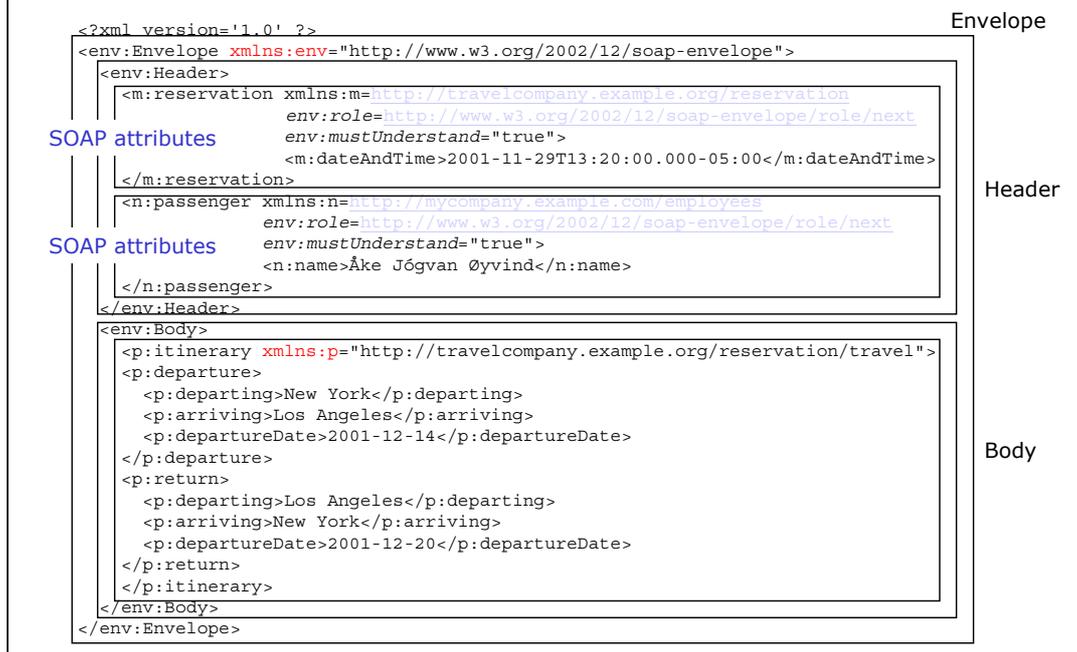


- SOAP messages consist of
 - Envelope: top element of XML message (required)
 - Header: general information on message such as security (optional)
 - Body: data exchanged (required)
- Header
 - elements are application-specific
 - may be processed and changed by intermediaries or recipient
- Body
 - elements are application-specific
 - processed by recipient only



The basic structure of a SOAP message consists of a header and a body, both of which the contents are application specific, i.e. not defined by SOAP. The differentiation is made to distinguish information that is to be processed by all intermediaries (Header) and information that is to be processed at the final destination (Body).

Example of SOAP Message

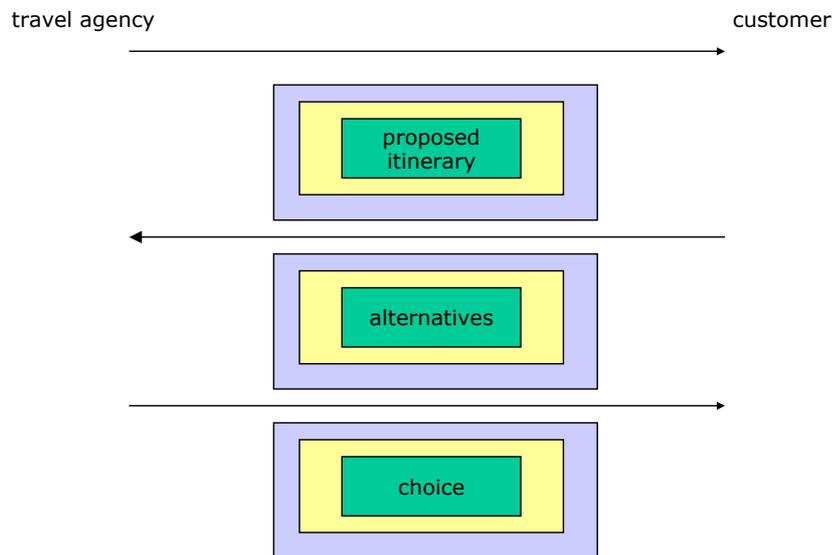


We highlight some specificities of the SOAP message model in this example. The envelope is the mandatory top-level element of any SOAP message.

The header relates to information on authentication, transaction management, payment etc. needed for processing the message and during transport. Elements in the header contain SOAP-specific attributes (recognizable as belonging to the env namespace) which are used to determine which type of processing is expected by intermediaries and endpoints. This example shows how the header is used in order to transmit certain transactional properties that are required for the service invocation.

The body contains the request, which consists of a travel itinerary in this case. In other words, this message is not corresponding to a method invocation but to a simple message transfer.

Conversational Message Exchanges in SOAP



©2004-2005, Karl Aberer & J.P. Martin-Flatin

14

Having simple informational messages as in the previous example, using SOAP, one can implement conversational message exchanges such as illustrated in this example. Such exchanges can for example be used to model a negotiation process between a seller and a customer.

SOAP RPC

- Encapsulate RPCs into SOAP messages
 - Procedure name and arguments
 - Response (return value)
 - Processing instructions
- SOAP RPC interactions do not exactly correspond to the classical RPC concept
 - May be implemented synchronously or asynchronously
 - depends on client

SOAP also standardizes the way request/response interactions are to be represented within a SOAP message. Note that this type of interaction does not exactly correspond to the classical concept of RPC, as it is implemented asynchronously (no common state). But it is called RPC in the SOAP standard and is supposed to be used to access services in an RPC style.

Example of SOAP RPC Request Message

```
<?xml version='1.0' ?>
<env:Envelope xmlns:env="http://www.w3.org/2002/12/soap-envelope" > transaction information
  <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.example.org/"
      >
      <m:reservation xmlns:m="http://travelcompany.example.org/reservation">
        <m:code>FT35ZBQ</m:code>
      </m:reservation>
      <o:creditCard xmlns:o="http://mycompany.example.com/financial">
        <n:name xmlns:n="http://mycompany.example.com/employees">
          Áke Jógvan Øyvind </n:name>
          <o:number>123456789099999</o:number>
          <o:expiration>2005-02</o:expiration>
        </o:creditCard>
      </m:chargeReservation>
    </env:Body>
  </env:Envelope>
```

TID

method invocation

parameter 1

parameter 2

©2004-2005, Karl Aberer & J.P. Martin-Flatin

16

The example shows how a request is represented. The (application-specific) method name (`chargeReservations`) occurs as top element in the message Body. The (application-specific) parameters are then to be included as sub-elements of the method element. SOAP also specifies how to encode various data types. We will not go into these details of the standard in the following.

The Header of the message can be used to include information that is necessary to properly process the request: in this example it is used to state that the message is to be performed as part of a transaction, with a given TID. Note that also the encoding of this processing information is not standardized by SOAP but by the applications, resp. systems, using the SOAP message format.

Example of SOAP RPC Response Message

```
<?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>
```

method result

output params

Similarly as the request also the response is encoded. In the response the method result is represented by an application specific top element in the message Body, which again includes one or more output parameters as sub-elements. In order to model functions SOAP provides an alternative representation of response messages allowing to distinguish a function result value as a special element.

SOAP Header Processing Model

- Elements in the Header may carry SOAP-specific attributes controlling the message processing
 - attributes from namespace <http://www.w3.org/2002/12/soap-envelope>
 - role, mustUnderstand, relay
- "role" attribute
 - if processing node matches role in header, it must process the header
 - special role "next": receiving node must be capable of processing header
 - special role "ultimateReceiver": receiving node must be capable of processing body
- "mustUnderstand" attribute
 - processing of header information is mandatory
- "relay" attribute
 - header block must be relayed if it is not processed

We now explain in more detail the meaning of the SOAP-specific attributes that are used in the elements occurring in the Header element. The role attribute is considered by the receiver of the message and concerns the capability of the receiver to process the (application-specific) information in the SOAP message. If its value is set to "next" this implies the every receiving node must be capable of processing the header, whereas the value "ultimateReceiver" implies the capability to also process the message body.

The "mustUnderstand" attributes can be used to control the processing of the header. Note that being capable of processing the header does not necessarily imply that the header will be processed. The "relay" attribute can in the case processing is not done, require to forward the message to another SOAP node.

Protocol Binding

- Bindings to different protocols possible: HTTP, BEEP, SMTP, etc.
- Different HTTP methods: HTTP POST, HTTP GET
 - Example using HTTP POST

```
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>
```

HTTP POST
request

```
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>
```

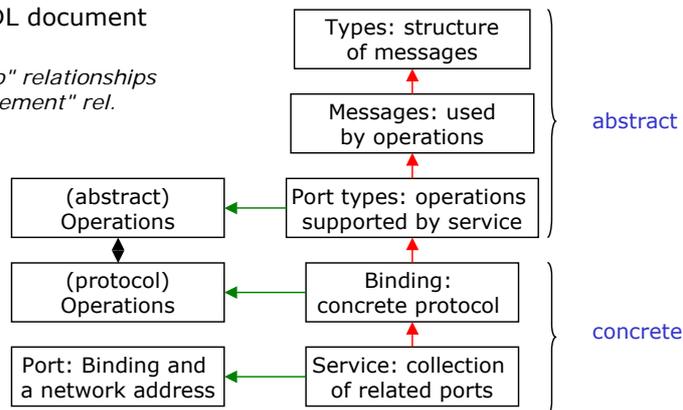
HTTP response

SOAP also standardizes the binding of SOAP messages to specific transfer protocols, in particular to HTTP. Essentially it prescribes of how the transfer protocols are to be used to perform SOAP message exchanges. An example of binding for HTTP is given in this example. Specifically, this type of binding requires the use of HTTP POST to transmit a request, and the use of HTTP response to transmit the response associated with the request.

7.3 WSDL – Web Service Description Language

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

Red arrows = "refersTo" relationships
Green arrows = "subElement" rel.



©2004-2005, Karl Aberer & J.P. Martin-Flatin

20

WSDL describes Web Services in terms of the services offered and the *endpoints* that offer the services. Using the WSDL specification of a Web Service, a client is able to (i) construct the necessary SOAP messages in order to access the service, (ii) send the message to the required network location, and (iii) correctly interpret the responses.

WSDL distinguishes the abstract and concrete specification of a service. The abstract part specifies the type of data used in the parameters, the types of messages exchanged during an operation, and the operations themselves, which might require the exchange of multiple messages. Since a Web Service might consist of a set of operations supported (e.g. a travel reservation service might offer operations to check itineraries, book flights and payment) multiple operations can be bundled in so-called portTypes.

The concrete specification of the service concerns the protocol used and the necessary binding (e.g. to SOAP) and the network addresses where the service is offered (called ports). Again the same service may be offered at multiple physical sites, therefore a set of ports can be specified to define the service.

This organization is also reflected in the structure of a WSDL document. The vertical arrows in the figure correspond to references made within a WSDL document through a "name" attribute (thus they are "refersTo" relationship) whereas the horizontal arrows correspond to sub-element relationships in the WSDL document.

How to Define a WSDL Service?

1. Define in XML Schema the types used when invoking the service: T1, T2 etc.
2. Define messages by using these types, e.g.
 - message m1 has type T1
 - message m2 has type T2 etc.
3. Define a service S as a port type P that consists of one or more operations; each operation is implemented by the exchange of messages
 - port type P 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, e.g. by inserting the message as body of SOAP messages
5. Service S is provided with binding B at the following URIs (called ports)

To better understand the role of the different parts of a WSDL document, we sketch here the typical steps that would be taken in completely defining a Web Service using WSDL.

Example: Overall Document Structure

<pre><?xml version="1.0"> <definitions name="StockQuote"> <types> <schema> definition of types in XML Schema </schema> </types> <message name="GetTradePriceInput"> definition of a message... </message> <portType name="StockQuotePortType"> <operation name="GetLastTradePrice"> definition of an operation </operation> </portType> <binding name="StockQuoteSoapBinding"> definition of a binding </binding> <service name="StockQuoteService"> <port name="StockQuotePort"> definition of a port </port> </service> </definitions></pre>	<p>Types</p> <p>Messages</p> <p>Port Types</p> <p>Binding</p> <p>Service</p>
--	--

22

This is an overview of the syntactical representation of a service in WSDL. For each of the main parts (data, messages, portTypes, Service) a separate top element is used (outer boxes). Within this subelements represent collections of specifications, such as the set of operations belonging to a portType. These are indicated by the inner boxes.

Example: Definition of Types

```
<?xml version="1.0"?>
<schema targetNamespace="http://example.com/stockquote/schemas"
        xmlns="http://www.w3.org/2000/10/XMLSchema">

  <element name="TradePriceRequest">
    <complexType>
      <all>
        <element name="tickerSymbol" type="string"/>
      </all>
    </complexType>
  </element>
  <element name="TradePrice">
    <complexType>
      <all>
        <element name="price" type="float"/>
      </all>
    </complexType>
  </element>
</schema>
```

This example shows the type definition of a WSDL specification. The types are defined using the XML Schema language. In this example two data types are defined as complex types. XML document (fragments) corresponding to these types will be used as parameters of the Web Service.

Example: Definition of Messages and Port Types

```
<?xml version="1.0"?>
<definitions name="StockQuote"

  targetNamespace="http://example.com/stockquote/definitions"
  xmlns:tns="http://example.com/stockquote/definitions"
  xmlns:xsd1="http://example.com/stockquote/schemas"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns="http://schemas.xmlsoap.org/wsdl/"

  <import namespace="http://example.com/stockquote/schemas"
    location="http://example.com/stockquote/stockquote.xsd"/>

  <message name="GetLastTradePriceInput">
    <part name="body" element="xsd1:TradePriceRequest"/>
  </message>
  <message name="GetLastTradePriceOutput">
    <part name="body" element="xsd1:TradePrice"/>
  </message>
  <portType name="StockQuotePortType">
    <operation name="GetLastTradePrice">
      <input message="tns:GetLastTradePriceInput"/>
      <output message="tns:GetLastTradePriceOutput"/>
    </operation>
  </portType>
</definitions>
```

type

type

Operation uses these messages

©2004-2005, Karl Aberer & J.P. Martin-Flatin

24

This fragment of a WSDL specification refers to the type definitions of the previous slide. Two messages are defined. The part elements of a message allow to associate with the message one or more types that have been defined before in the type specification. Each part obtains an individual name and refers to a type.

A portType consists of one or more operations. Each operation requires two messages, one for the request (input) and one for the response. The messages are referred to by their names.

This example illustrates of how a WSDL definition can be decomposed into multiple documents: the type definitions from the previous slides are imported into the document using the import element (rather directly inlined into the document, which also would be possible).

Note that all element and attribute names in this example are WSDL-specific, i.e. defined in the WSDL schema (which is given as an XML Schema)

Example: Definition of Binding and Service

```
<?xml version="1.0"?>
<definitions name="StockQuote"

targetNamespace="http://example.com/stockquote/service"
  xmlns:tns="http://example.com/stockquote/service"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:defs="http://example.com/stockquote/definitions"
  xmlns="http://schemas.xmlsoap.org/wsdl/">

  <import namespace="http://example.com/stockquote/definitions"
    location="http://example.com/stockquote/stockquote.wsdl"/>

  <binding name="StockQuoteSoapBinding" type="defs:StockQuotePortType">
    <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
    <operation name="GetLastTradePrice">
      <soap:operation soapAction="http://example.com/GetLastTradePrice"/>
      <input>
        <soap:body use="literal"/>
      </input>
      <output>
        <soap:body use="literal"/>
      </output>
    </operation>
  </binding>

  <service name="StockQuoteService">
    <documentation>My first service</documentation>
    <port name="StockQuotePort" binding="tns:StockQuoteBinding">
      <soap:address location="http://example.com/stockquote"/>
    </port>
  </service>
</definitions>
```

the abstract operation GetLastTradePrice of port type StockQuotePortType is implemented by these SOAP messages

Binding provided at this URI

©2004-2005, Karl Aberer & J.P. Martin-Flatin

25

This WSDL fragment completes the example. It refers to the specification on the previous slide. In the binding element it binds the port type GetLastTradePrice (which is the abstract service) to a concrete protocol, namely the SOAP protocol and thus defines the message format. In the service part the binding StockQuoteBinding (which is the abstract service definition together with a concrete transport protocol) is bound to a specific access point, given as URL, resulting in a port. This single port then constitutes the concrete service.

We do not detail the rules of how the binding between the abstract operation specification and its corresponding SOAP operations is established. Essentially, using the syntax given above, the input/output messages are mapped to SOAP request/response messages, and the SOAP message body is derived from the XML types associated with the messages in the abstract specification.

Port Types

- WSDL supports 4 message patterns that an endpoint (= service provider) can support for an operation
 - *one-way*: message is sent to service provider, no response expected
 - *request-response*: message is sent to service provider, response expected
 - *solicit-response*: service provider sends message, response expected
 - *notification*: service provider sends message, no response expected
- Message patterns are distinguished by the use of input/output elements
 - one way:

```
<wsdl:definitions .... > <wsdl:portType .... > *
  <wsdl:operation name="nmtoken">
    <wsdl:input name="nmtoken"? message="qname"/>
  </wsdl:operation>
</wsdl:portType >
</wsdl:definitions>
```
 - request-response:

```
<wsdl:definitions .... >
  <wsdl:portType .... > *
    <wsdl:operation name="nmtoken" parameterOrder="nmtokens">
      <wsdl:input name="nmtoken"? message="qname"/>
      <wsdl:output name="nmtoken"? message="qname"/>
      <wsdl:fault name="nmtoken" message="qname"/>*
    </wsdl:operation>
  </wsdl:portType>
</wsdl:definitions>
```

WSDL foresees four different types of operations, corresponding to different message exchange patterns. These patterns are considered as being the most frequently used in practice. These different patterns are distinguished in the specification of the operation by the different use of the input/output elements, as illustrated in the example.

7.4 UDDI – Universal Description, Discovery and Integration

- Standard for describing, publishing and finding Web Services
 - Uses XML-based description files for services
 - Still evolving
 - Main weak point of Web services to date
- 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 about the organization
 - Business services: business level description of service
 - Binding templates: access point + references to tModels
 - tModel: technical model = abstract definition of a Web Service (indep. of WSDL)

©2004-2005, Karl Aberer & J.P. Martin-Flatin

27

The UDDI standard contains a number of different components for describing organizations, classifying them according to their general activities and allowing them to offer registered Web Services. UDDI also standardizes an access protocol for accessing a UDDI repository. First UDDI repositories are being made publicly or commercially available. These can also be accessed through Web interfaces.

A business is described in UDDI by an XML document. UDDI introduces an abstract model to specify Web Services, the tModel. This seems to be redundant with WSDL, but the purpose is different: UDDI foresees that different abstract service description mechanisms might be used in future. The tModel provides a common framework to include these into UDDI registry entries.

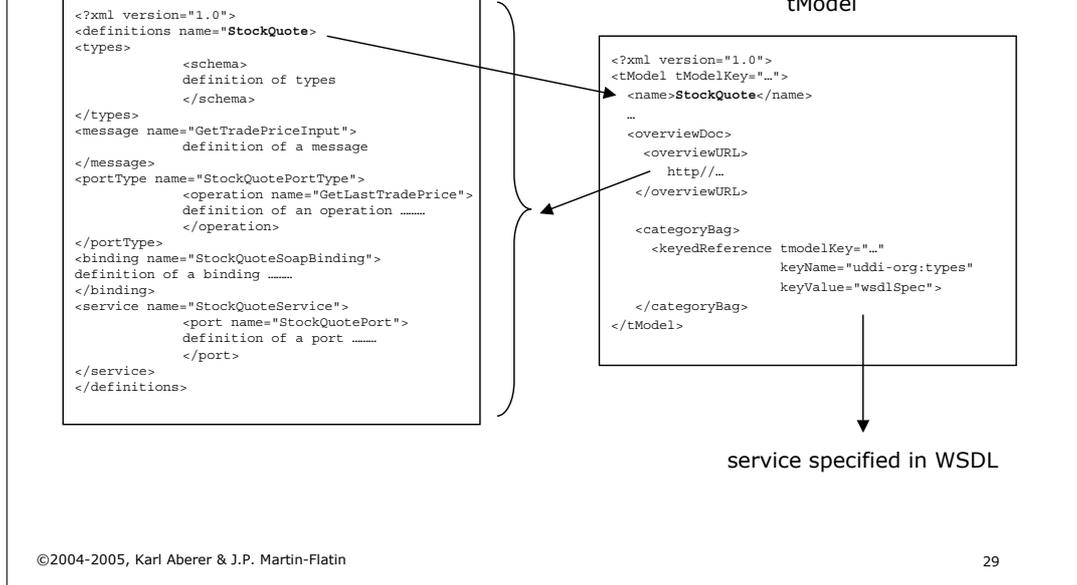
Registering a WSDL Service in UDDI

1. Register a business
 2. Register the abstract service definition (tModel)
 3. Register the service implementation definition (BusinessService)
- Step 1: Register a business
(see demo at <https://uddi.ibm.com/testregistry/>)

The screenshot shows the IBM UDDI Business Test Registry interface. The main heading is "UDDI Business Test Registry" with the subtitle "Universal Description, Discovery, and Integration". The primary action is "Add a Business". A text box prompts the user to "Enter the name of your business" with a "Name*" input field. A "Language*" dropdown menu is set to "English". Below the input fields are "Cancel" and "Continue" buttons. The left sidebar contains navigation links for "UDDI Business Registry", "Logout", "Publish", "Find", and "Account". The footer includes "©2004-2005, Karl Aberer & J.P. Martin" and "About IBM | Privacy | Legal | Contact".

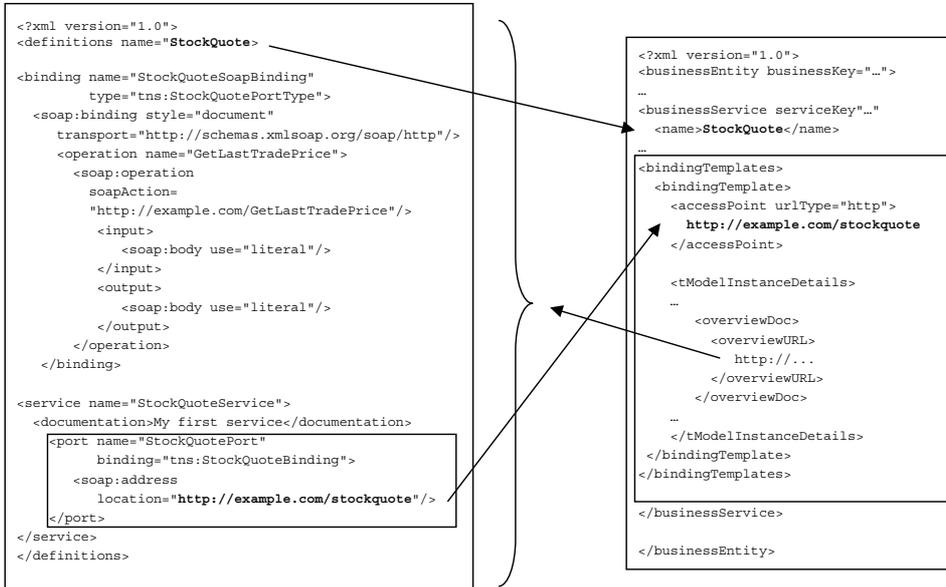
Registration of service can be performed interactively at UDDI repositories. The registration of a business requires to capture basic information (like address, areas of activity). We will give a demo of how this is interactively performed.

Step 2: Registering an Abstract WSDL Service Definition



Registering a service requires two steps: the registration of the abstract description and the registration of the service implementation-related information. The abstract definition is captured by the so-called tModel (another XML document type). It allows to provide the UDDI registry with the information on the service name (i.e. its identification) and its abstract specification, by referring to a WSDL document. The figure illustrates of how this association is syntactically represented in XML. UDDI provides for every service representation standard a model for registering the description, though WSDL is apparently the only relevant one at the given time.

Step 3: Registering a Service Implementation



©2004-2005, Karl Aberer & J.P. Martin-Flatin

30

Similarly, the concrete implementation of the service is registered (as part of the business service description) by referring to a WSDL document. For the registration of the implementation, in particular the physical address at which the service is accessed, which is found in the port definition in the WSDL specification, is included in the UDDI service description as shown in the example. Again the UDDI service description then refers for the details of the service binding to the WSDL document.

Demo

<https://uddi.ibm.com/testregistry/registry.html>

Summary

- Web Services are becoming the predominant mechanism for distributed computing on the WWW
- They implement a service stack comparable to earlier distributed computing solutions such as CORBA, but are fully based on Web standards and therefore can take advantage of the Web infrastructure
- The current standards include SOAP for communication, WSDL for service specification, and UDDI for service discovery
- Higher level standards for service composition and semantic description of Web Services are currently emerging

References

- Standard documents
 - <http://www.w3.org/2002/ws/>
 - <http://www.w3.org/TR/2002/CR-soap12-part0-20021219/> (SOAP primer)
 - <http://www.w3.org/TR/SOAP/>
 - <http://www.w3.org/TR/wsdl>
 - <http://www.uddi.org>
- Articles
 - <http://www2002.org/CDROM/alternate/395/> (Comparison to CORBA)