Lesson 4 – Web Service Interface Definition (Part I)

Service Oriented Architectures Security

Module 1 - Basic technologies

Unit 3 – WSDL

Ernesto Damiani

Università di Milano

Interface Definition Languages (1)

- IDLs (Interface Definition Languages) are typically used in implicit communication approaches, where communication primitives are hidden behind procedure or method calls and, thus, communicating implies knowing the interface of the procedure/method at the other side
  - a language for declaring interfaces (used to publish interfaces and by the compiler and linker to decide what to do when a call to a remote procedure/method takes place)
  - a mapping between data representations in the language to an intermediate representation (for marshalling)
  - a way of serializing the data for transmission
  - compilers to turn the interface definitions into modules that can be linked and compiled into other software modules
    - most IDLs are character based (data is represented in ASCII)

Interface Definition Languages (2)

- For implicit communication, the intermediate representation is strongly tied to how interfaces are defined for software modules at the programming language level
- The IDL allows to define each service in terms of their names, and input and output parameters (plus maybe other relevant aspects)

Interface Definition Languages (3)

- All RPC systems have a language that allows to describe services in an abstract manner (independent of the programming language used)
  - This language has the generic name of IDL (e.g., the IDL of SUN RPC is called XDR)
- An interface compiler is then used to generate the stubs for clients and servers (rpcgen in SUN RPC)
  - Rpcgen generates procedure headings that the programmer can then use to fill out the details of the implementation

Interface Definition Languages (4)

- Given an IDL specification, the interface compiler performs a variety of tasks
  - It generates the client stub procedure for each procedure signature in the interface. The stub will be then compiled and linked with the client code
  - It generates a server stub. It can also create a server main, with the stub and the dispatcher compiled and linked into it. This code can then be extended by the designer by writing the implementation of the procedures
  - It might generate a *.h file for importing the interface and all the necessary constants and types

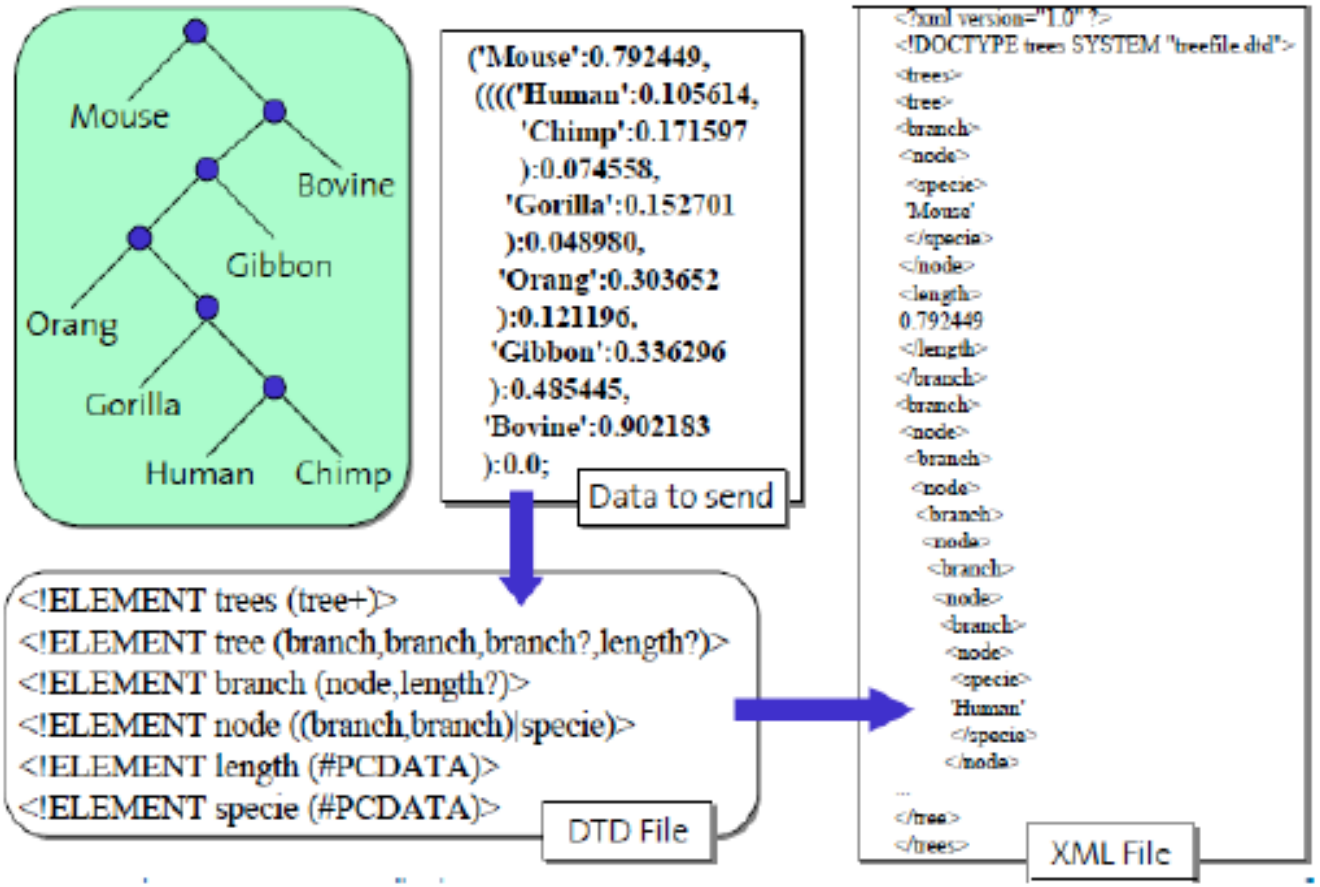
XML reminder (1)

- The goal of XML is to provide a standardized way to specify data structures so that when data is exchanged, it is possible to understand what has been sent
  - The DTD (Document Type Definition) specifies how the data structure is described: processing instructions, declarations, comments, and elements
  - Using the DTD, the XML document can be correctly interpreted by a program by simply parsing the document using the grammar provided by the DTD
  - The idea is similar to IDL except that instead of defining parameters as combinations of standard types, a DTD describes arbitrary documents as semi-structured data

XML reminder (2)

- Using XML is possible to exchange data through HTTP and Web servers and process the data automatically
  - The use of XML reduces the universality of the browser since now a browser needs additional programs to deal with specific markup languages developed using XML (somewhat similar to plug-ins but more encompassing in terms of functionality)
  - However, this is not much of a problem since the browser is for humans while XML is for automated processing
- XML can be used as the intermediate language for marshalling/serializing arguments when invoking services across the Internet

XML reminder (3)



XML Schema (1)

- A different problem related to accessing EAI systems through a web interface is the representation of relational data
  - If HTML is used, the data is formatted for presentation, not for processing
  - If XML and DTDs are used, then the structured is better suited for processing but the processing is ad-hoc (one can define any DTD one wants)
- XML Schema has been proposed to allow database like query processing over XML documents.
  - XML Schema is a data definition language for XML documents that allows to treat them as relational data in a standardized manner

XML Schema (2)

- What is different between XML Schema and DTDs?
- XML Schema
  - uses the same syntax as XML (DTDs have a different syntax)
  - provides a wider set of types (similar to those in SQL)
  - allows to define complex types from the basic types
  - supports key and referential integrity constraints
  - can be used by query languages (XQuery, for instance) to parse XML documents and treat them as relational data
  - can be used to specify the data model used by a Web service interface

WSDL: the notion (1)

- WSDL (Web Services Description Language) can be best understood when we approach it as an XML version of an IDL that also covers the aspects related to integration through the Internet and the added complexity of Web services
- A conventional IDL does not include information such as:
  - location of the service (implicit in the platform and found through static or dynamic binding)
  - different bindings (typically an IDL is bound to a transport protocol)
  - sets of operations (since an interface defines a single access point and there is no such a thing as a sequence of operations involved in the same service)

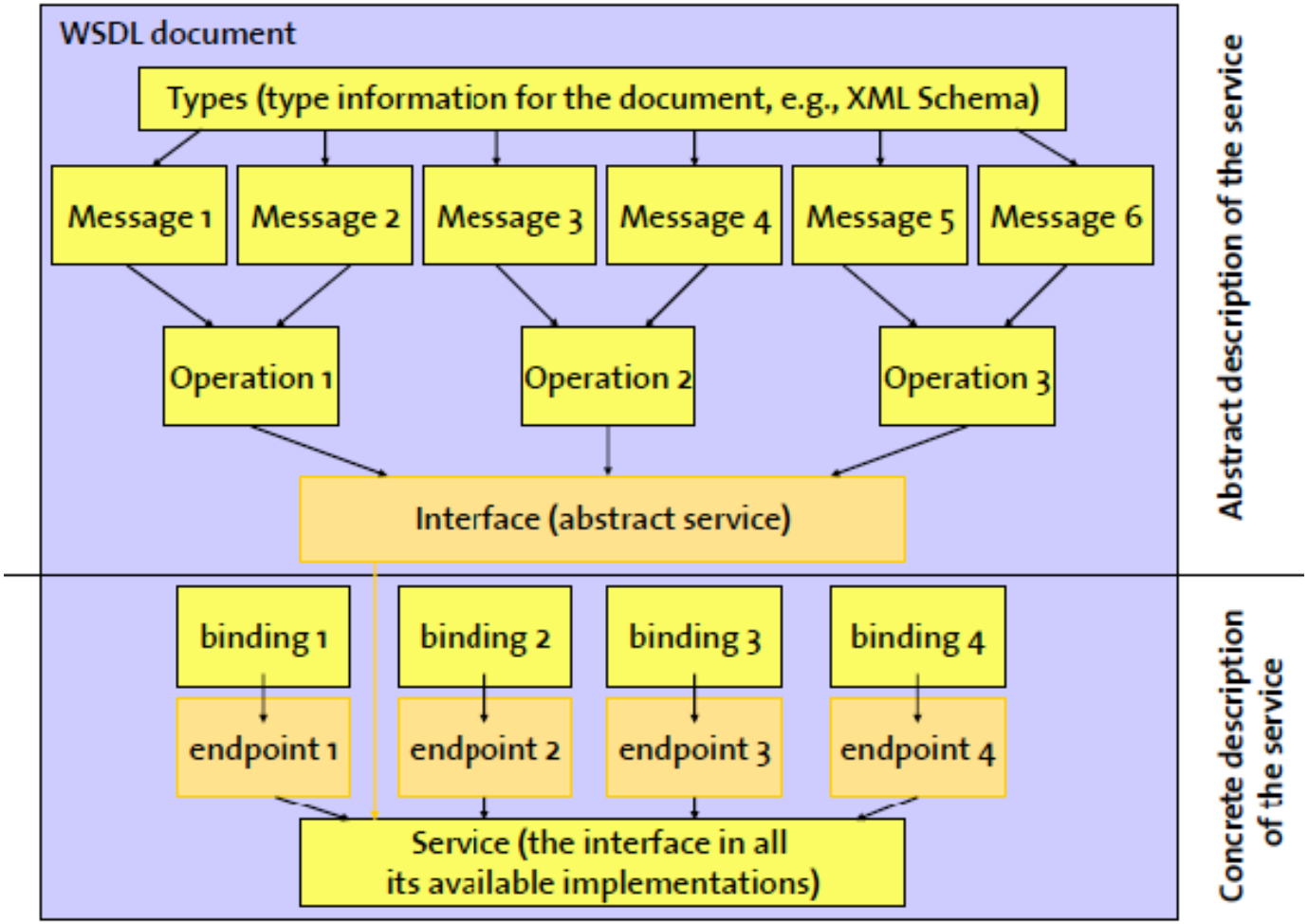
WSDL: the notion (2)

- An IDL in conventional middleware and enterprise application integration platforms has several purposes:
  - description of the interfaces of the services provided (e.g., RPC)
  - serve as an intermediate representation for bridging heterogeneity by providing a mapping of the native data types to the intermediate representation associated to the IDL in question
  - serve as the basis for development through an IDL compiler that produces stubs and libraries that can be used to develop the application

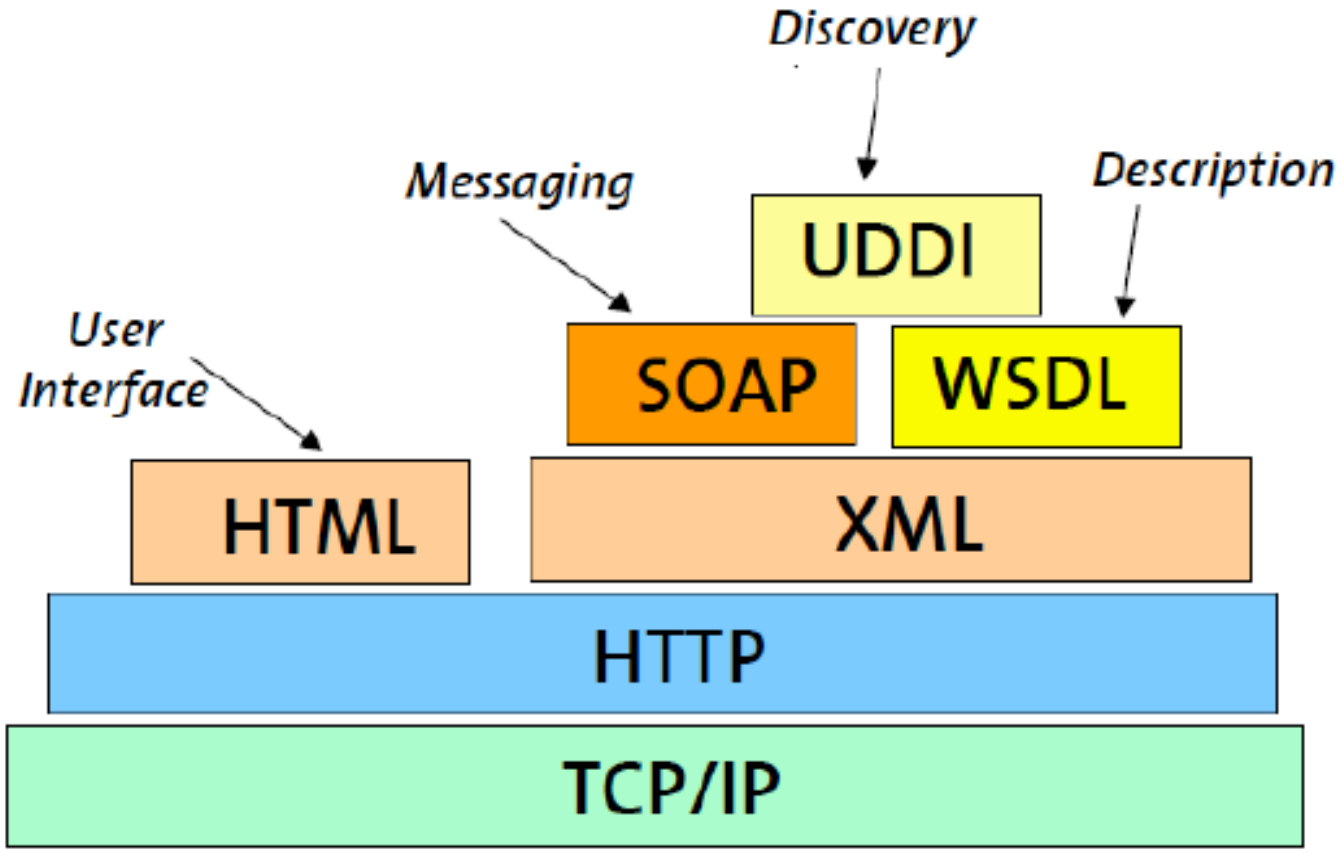
Introduction to WSDL

- The WSDL specification is in Version v2.0 (June 2007)
  - it discusses how to describe the different parts that comprise a Web service interface the type system used to describe the service data model (XML Schema)
  - the messages involved in the interaction with the service
  - the individual operations composed of 4 possible message exchange patterns
  - the sets of operations that constitute a service
  - the mapping to a transport protocol for the messages
  - the location where the service provider resides
  - groups of locations that can be used to access the same service
- It also includes a specification indicating how to bind WSDL to the SOAP, HTTP (POST/GET) and MIME protocols

Elements of WSDL 2.0



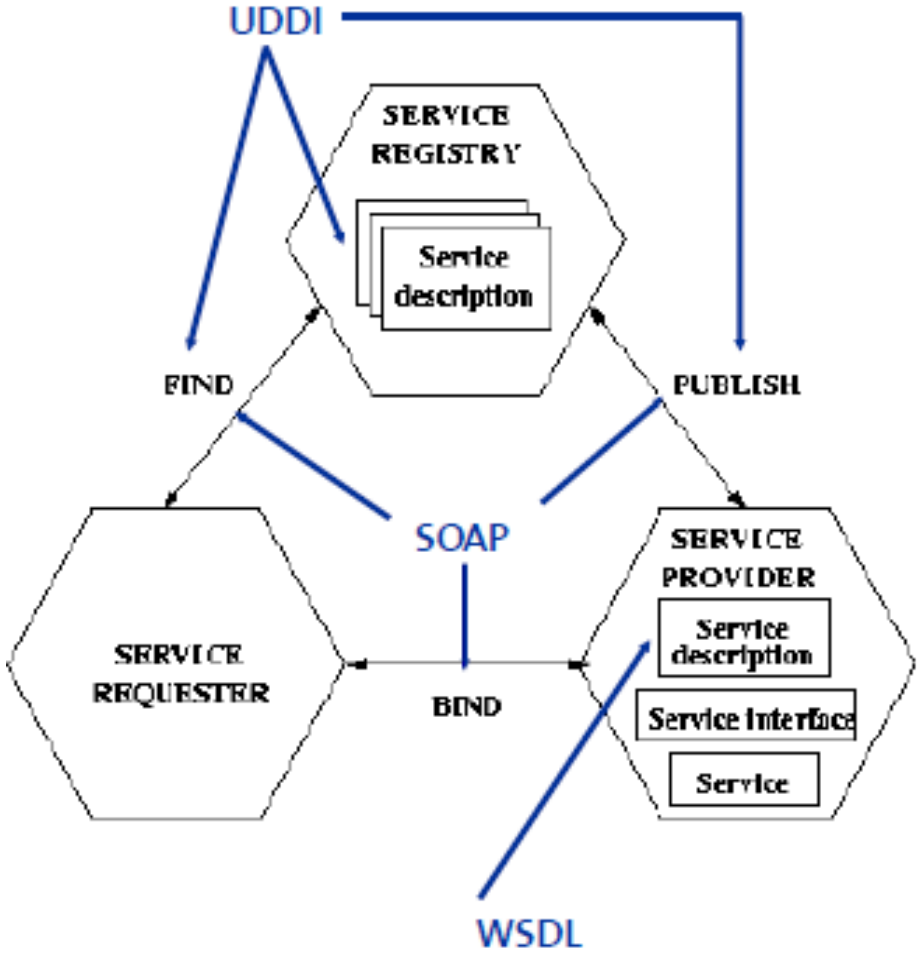
Layering


Discovery
Messaging
Description
UDDI
User Interface
SOAP
WSDL
HTML
XML
HTTP
TCP/IP

The role of WDSL/UDDI (1)

- Once it is possible to interact with any service provider using the standard SOAP protocol, it is still necessary to:
  - describe the services (WSDL)
  - discover the services (UDDI, Universal Description, Discovery and Integration)

The role of WDSL/UDDI (2)



WSDL data types

- The types in WSDL are used to specify the contents of the messages (normal messages and fault messages) that will be exchanged as part of the interactions with Web services
  - the type system is typically based on XML Schema (structures and data types)
  - support is mandatory for all WSDL processors
  - an extensibility element can be used to define a schema other than XML Schema

WSDL data types: example

PURCHASE ORDER TYPE

```
<element name="PO" type="tns:POType"/>
<complexType name="POType">
  <all>
    <element name="id" type="string"/>
    <element name="name" type="string"/>
    <element name="items">
      <complexType>
        <all>
          <element name="item" type="tns:Item" minOccurs="0" maxOccurs="unbounded"/>
        </all>
      </complexType>
    </element>
  </all>
</complexType>
```

ITEM TYPE

```
<complexType name="Item">
  <all>
    <element name="quantity" type="int"/>
    <element name="product" type="string"/>
  </all>
</complexType>
```

INVOICE TYPE

```
<element name="Invoice" type="tns:InvoiceType"/>
<complexType name="InvoiceType">
  <all>
    <element name="id" type="string"/>
  </all>
</complexType>
```

From Web Services Description Language (WSDL) 1.1 W3C Note 15 March 2001

Messages and faults (1)

- Messages have a name that identifies them throughout the XML document. They are divided into parts, each of them being a data structure represented in XML. Each part must have a type (basic or complex types, previously declared in the WSDL document)
  - A WSDL message element matches the contents of the body of a SOAP message. By looking at the types and looking at the message, it is possible to build a SOAP message that matches the WSDL description (and this can be done automatically since the description is XML based and the types also supported by SOAP)
  - A message does not define any form of interaction, it is just a message

Messages and faults (2)

- In WSDL 1.0, the structure of a “message” was explicitly defined, listing all of its parts
- In WSDL 2.0, a “message reference component” is defined as part of an operation and contains three elements
  - message label (indicating the message pattern used for the message)
  - direction (whether it is an inbound or outbound message)
  - message element (the actual contents of the message expressed in terms of the types previously defined)

Messages and faults (3)

- Faults are a special kind of message used to report errors

```
<message name="PO">                                          1.0
   <part name="po" element="tns:PO"/>
   <part name="invoice" element="tns:Invoice"/>
</message>
```

Operations

- In WSDL 2.0, an operation is a set of messages and faults. The sequencing and number of messages in the operation is determined by the message exchange pattern
  - The style of an operation distinguishes between RPC-like behavior, document oriented message exchange or (in 2.0) set-and get-of attributes
  - Operations can be annotated with features and properties (e.g., reliability, security, routing)

ONE-WAY:

```
<wsdl:operation name="Purchase">
  <wsdl:input name="Order" message="PO"/>
</wsdl:operation>
```

REQUEST-RESPONSE:

```
<wsdl:operation name="Purchase">
  <wsdl:input name="Order" message="PO"/>
  <wsdl:output name="Confirm" message="Conf"/>
  <wsdl:fault name="Error" message="POError"/>
</wsdl:operation>
```

Interfaces (1)

- An interface corresponds to the abstract definition of a Web service (abstract because it does not specify any information about where the service resides or what protocols are used to invoke the Web service)
- The interface is simply a list of operations that can be used in that Web service
  - Operations are not defined by themselves but only as part of an interface

Interfaces (2)

From Web Services Description Language (WSDL) 1.1 W3C Note 15 March 2001

```
<message name="m1">
  <part name="body" element="tns:GetCompanyInfo"/>
</message>

<message name="m2">
 <part name="body" element="tns:GetCompanyInfoResult"/>
 <part name="docs" type="xsd:string"/>
 <part name="logo" type="tns:ArrayOfBinary"/>
</message>

<portType name="pt1">
  <operation name="GetCompanyInfo">
       <input message="m1"/>
       <output message="m2"/>
  </operation>
</portType>
```

Bindings (1)

- A binding defines message formats and protocol details for the operations and messages of a given Port Type (end point in the new spec)
- A binding corresponds to a single end point (obvious since it needs to refer to the operations and messages of the end point)

Bindings (2)

- An end point can have several bindings (thereby providing several access channels to the same abstract service)
  - The binding is extensible with elements that allow to specify mappings of the messages and operations to any format or transport protocol. In this way, WSDL is not protocol specific

End points

- An end point specifies the address of a binding, i.e., how to access the service using a particular protocol and format
- End points can only specify one address and they should not contain any binding information
- The end point is often specified as part of a service rather than on its own

Services

- Services group a collections of ports together and therefore become the complete definition of the service as seen by the outside: a service supports several protocols (it has several bindings)
  - access to the service under a given protocol is through a particular address (specified in the ports of each binding)
  - operations and messages to exchange are defined in the End Point
  - ports that are part of the same service may not communicate with each other
  - ports that are part of the same service are considered as alternatives all of them with the same behavior (determined by the End Point) but reachable through different protocols

Bindings and ports: example

```
<binding name="b1" type="tns:pt1">
    <operation name="GetCompanyInfo">
        <soap:operation soapAction="http://example.com/GetCompanyInfo"/>
            <input>
              <soap:body use="literal"/>
            </input>
            <output>
              <mime:multipartRelated>
                <mime:part>
                  <soap:body parts="body" use="literal"/>
                </mime:part>
                <mime:part>
                  <mime:content part="docs" type="text/html"/>
                </mime:part>
                <mime:part>
                  <mime:content part="logo" type="image/gif"/>
                  <mime:content part="logo" type="image/jpeg"/>
                </mime:part>
              </mime:multipartRelated>
            </output>
    </operation>
</binding>
<service name="CompanyInfoService">
        <port name="CompanyInfoPort"binding="tns:b1">
                <soap:address location="http://example.com/companyinfo"/>
        </port>
</service>
```

From Web Services Description Language (WSDL) 1.1 W3C Note 15 March 2001

RPC vs. REST style

- The style of a SOAP message controls the format of the <soap:Body> element:

▪ **RPC** style, an extra child element of the Body is added to identify the method to be called. Parameters are listed inside this one.	▪ **Document** style, the Body contains an arbitrary XML document.
<soap:Body> <tns:ConfirmOrder xmlns:tns="http://my.package/"> <number xsi:type="xsd:integer">1234</number> <confirm xsi:type="xsd:boolean">true</confirm> </tns:Method> </soap:Body>	<soap:Body> <tns:order number="00001234"> Purchase Order Confirmation <tns:status>Confirmed<tns:status> … </tns:order> </soap:Body>
ConfirmOrder(number,confirm);	*Send(OrderDocument);*