Web Service Standards: Overview

Contents

1 Interface Description **1**
- 1.1 WSDL (Web Services Description Language) . . . 1
  - 1.1.1 WSDL Document Example . . . 2

2 Service Discovery **4**
- 2.1 UDDI: (Universal Description, Discovery, and Integration) . . . 4
- 2.2 WS-Discovery (Web Services Dynamic Discovery) . . . 4
  - 2.2.1 Terms and definitions . . . 5
  - 2.2.2 Example Probe sent multicast in ad hoc mode . . . 6
  - 2.2.3 Example ProbeMatch sent in response to Probe . . . 6

3 Service Activation and Messaging **7**
- 3.1 SOAP (Simple Object Activation Protocol) . . . 7

4 Service Orchestration **8**
- 4.1 WS-BPEL (Business Process Execution Language) . . . 8
  - 4.1.1 History . . . 8
  - 4.1.2 Orchestration vs Choreography . . . 8
  - 4.1.3 Example purchaseOrderProcess . . . 9

5 Service Choreography **12**
- 5.1 WS-CDL (Web Services Choreography Description Language) . . . 12
  - 5.1.1 WS-CDL model overview . . . 12
- 5.2 WSCI (Web Service Choreography Interface) . . . 13
- 5.3 WSCL (Web Service Conversation Language) . . . 13

1 Interface Description

1.1 WSDL (Web Services Description Language)

The **WSDL** is an XML-based IDL, which is used to describe the functionality of a web service. A WSDL document defines services as collections of network endpoints, or **ports**. In WSDL, the abstract definition of endpoints and messages is separated from their concrete network deployment or data format bindings. This allows the reuse of abstract definitions: **messages**, which are abstract descriptions of the data being exchanged, and **port types** which are abstract collections of **operations**. The concrete protocol and data format specifications for a particular port type constitutes a reusable **binding**. A port is defined by associating a network address with a reusable binding, and a collection of ports define a service. Hence, a WSDL document uses the following elements in the definition of network services:

- **Types**: a container for data type definitions using some type system (such as XSD); XML schema language (aka XSD) is used for this purpose
- **Message**: an abstract, typed definition of the data being communicated.

- **Operation**: an abstract description of an action supported by the service; SOAP actions and the way the message is encode – similar to functino/method call in PL
- **Port Type**: an abstract set of operations supported by one or more endpoints; *defines a web service, the operations that can be performed and the messages that are used to perform the operation*
- **Binding**: a concrete protocol and data format specification for a particular port type; specifies the interface and defines the SOAP binding style (RPC/Document) and transport (SOAP protocol); also defines operations
- **Endpoint (Port in 1.0)**: a single endpoint defined as a combination of a binding and a network address. typically represented by a simple **HTTP URL** string
- **Service**: a collection of related endpoints.

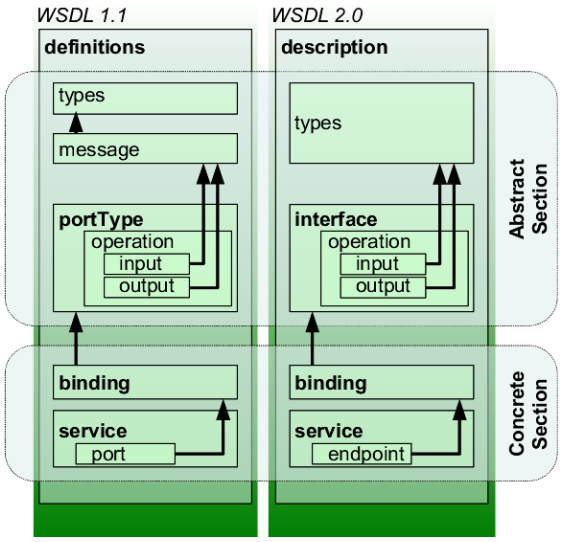

Figure 1: WSDL 1.0 vs 2.0

1.1.1 WSDL Document Example

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

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

    <types>
       <schema targetNamespace="http://example.com/stockquote.xsd"
              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>
</types>

<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>
<binding name="StockQuoteSoapBinding" type="tns: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>
```

2 Service Discovery

2.1 UDDI: (Universal Description, Discovery, and Integration)

UDDI, *which is obsolete now*, provides a systematic way to publish and discover (search) services. UDDI business registration consists of three components:

- **White Pages**: address, contact, and known identifiers;
- **Yellow Pages**: industrial categorizations based on standard taxonomies
- **Green Pages**: technical information about services exposed by the business.

White Pages White pages give information about the business supplying the service. This includes the name of the business and a description of the business – potentially in multiple languages. Using this information, it is possible to find a service about which some information is already known (for example, locating a service based on the provider's name).

Contact information for the business is also provided – for example the businesses address and phone number; and other information such as the Dun & Bradstreet Universal Numbering System number.

Yellow Pages Yellow pages provide a classification of the service or business, based on standard taxonomies. These include the Standard Industrial Classification (SIC), the North American Industry Classification System (NAICS), or the United Nations Standard Products and Services Code (UNSPSC) and geographic taxonomies.

Because a single business may provide a number of services, there may be several Yellow Pages (each describing a service) associated with one White Page (giving general information about the business).

Green Pages Green pages are used to describe how to access a Web Service, with information on the service bindings. Some of the information is related to the Web Service - such as the address of the service and the parameters, and references to specifications of interfaces. Other information is not related directly to the Web Service - this includes e-mail, FTP, CORBA and telephone details for the service. Because a Web Service may have multiple bindings (as defined in its WSDL description), a service may have multiple Green Pages, as each binding will need to be accessed differently.

2.2 WS-Discovery (Web Services Dynamic Discovery)

Web Services Dynamic Discovery (WS-Discovery) is a technical specification that defines a multicast discovery protocol to locate services on a local network. It operates over TCP and UDP port 3702 and uses IP multicast address 239.255.255.250. As the name suggests, the actual communication between nodes is done using web services standards, notably SOAP-over-UDP.

This specification defines a discovery protocol to locate services. The primary scenario for discovery is a client searching for one or more target services. The protocol defines two modes of operation, an ad hoc mode and a managed mode. In an ad hoc mode, to find a target service by the type of the target service, a scope in which the target service resides, or both, a client sends a

probe message to a multicast group; target services that match the probe send a response directly to the client. To locate a target service by name, a client sends a resolution request message to the same multicast group, and again, the target service that matches sends a response directly to the client.

To minimize the need for polling in an ad hoc network, when a target service joins the network, it sends an announcement message to the same multicast group. By listening to this multicast group, clients can detect newly available target services without repeated probing.

To scale to a large number of endpoints and to extend the reach of the protocol beyond the range of an ad hoc network, this specification defines a managed mode of operation and a multicast suppression behavior if a discovery proxy is available on the network. In managed mode, target services send unicast announcement messages to a discovery proxy and clients send unicast probe and resolve messages to a discovery proxy. To reduce multicast traffic, when a discovery proxy detects a probe or resolution request sent multicast on an ad hoc network, it sends an announcement for itself. By listening for these announcements, clients detect discovery proxies and switch to a managed mode of operation and send unicast probe and resolve messages directly to a discovery proxy. However, if a discovery proxy is unresponsive, clients revert to an ad hoc mode of operation.

To support networks with explicit network management services like DHCP, DNS, domain controllers, directories, etc., this specification acknowledges that clients and/or target services can be configured to behave differently than defined herein. For example, another specification may define a well-known DHCP record containing the address of a discovery proxy, and compliance with that specification may require client and target services to operate in a managed mode and send messages to this discovery proxy rather than to a multicast group. While the specific means of such configuration is beyond the scope of this specification, it is expected that any such configuration would allow clients and/or target services to migrate smoothly between carefully-managed and ad hoc networks.

2.2.1 Terms and definitions

Defined below are the basic definitions for the terms used in this specification.

- **Target Service**: An endpoint that makes itself available for discovery.
- **Client**: An endpoint that searches for Target Service(s).
- **Discovery Proxy**: An endpoint that facilitates discovery of Target Services by Clients.
- **Hello**: A message sent by a Target Service when it joins a network; this message contains key information for the Target Service. A Hello message is also sent by a Discovery Proxy to reduce multicast traffic on an ad hoc network; this message contains key information about the Discovery Proxy.
- **Bye**: A best-effort message sent by a Target Service when it leaves a network.
- **Probe**: A message sent by a Client searching for a Target Service by Type and/or Scope.
- **Resolve**: A message sent by a Client searching for a Target Service by name.
- **Type**: An identifier for a set of messages an endpoint sends and/or receives (e.g., a WSDL 1.1 portType, see [WSDL 1.1]).
- **Scope**: An extensibility point that allows Target Services to be organized into logical groups.
- **Metadata**: Information about the Target Service; includes, but is not limited to, transports and protocols a Target Service understands, Types it implements, and Scopes it is in.

- **Ad hoc Mode**: An operational mode of discovery in which the Hello, Bye, Probe and Resolve messages are sent multicast.
- **Managed Mode**: An operational mode of discovery in which the Hello, Bye, Probe and Resolve messages are sent unicast to a Discovery Proxy.
- **Ad hoc Network**: A network in which discovery is performed in an ad hoc mode.
- **Managed Network**: A network in which discovery is performed in a managed mode.

2.2.2 Example Probe sent multicast in ad hoc mode

```
<s:Envelope
   xmlns:a="http://www.w3.org/2005/08/addressing"
   xmlns:d="http://docs.oasis-open.org/ws-dd/ns/discovery/2009/01"
   xmlns:i="http://printer.example.org/2003/imaging"
   xmlns:s="http://www.w3.org/2003/05/soap-envelope" >
   <s:Header>
      <a:Action>
         http://docs.oasis-open.org/ws-dd/ns/discovery/2009/01/Probe
      </a:Action>
      <a:MessageID>
         urn:uuid:0a6dc791-2be6-4991-9af1-454778a1917a
      </a:MessageID>
      <a:To>urn:docs-oasis-open-org:ws-dd:ns:discovery:2009:01</a:To>
   </s:Header>
   <s:Body>
      <d:Probe>
         <d:Types>i:PrintBasic</d:Types>
         <d:Scopes
            MatchBy="http://oasis-open.org/ws-dd/ns/discovery/ldap" >
            ldap:///ou=engineering,o=examplecom,c=us
         </d:Scopes>
      </d:Probe>
   </s:Body>
</s:Envelope>
```

2.2.3 Example ProbeMatch sent in response to Probe

```
<s:Envelope
  xmlns:a="http://www.w3.org/2005/08/addressing"
  xmlns:d="http://docs.oasis-open.org/ws-dd/ns/discovery/2009/01"
  xmlns:i="http://printer.example.org/2003/imaging"
  xmlns:s="http://www.w3.org/2003/05/soap-envelope" >
  <s:Header>
    <a:Action>
      http://docs.oasis-open.org/ws-dd/ns/discovery/2009/01/ProbeMatches
    </a:Action>
    <a:MessageID>
      urn:uuid:e32e6863-ea5e-4ee4-997e-69539d1ff2cc
    </a:MessageID>
    <a:RelatesTo>
```

```
      urn:uuid:0a6dc791-2be6-4991-9af1-454778a1917a
    </a:RelatesTo>
    <a:To>
    http://www.w3.org/2005/08/addressing/anonymous
    </a:To>
    <d:AppSequence InstanceId="1077004800" MessageNumber="2" />
  </s:Header>
  <s:Body>
    <d:ProbeMatches>
      <d:ProbeMatch>
        <a:EndpointReference>
          <a:Address>
            urn:uuid:98190dc2-0890-4ef8-ac9a-5940995e6119
          </a:Address>
        </a:EndpointReference>
        <d:Types>i:PrintBasic i:PrintAdvanced</d:Types>
        <d:Scopes>
          ldap:///ou=engineering,o=examplecom,c=us
          ldap:///ou=floor1,ou=b42,ou=anytown,o=examplecom,c=us
          http://itdept/imaging/deployment/2004-12-04
        </d:Scopes>
        <d:XAddrs>http://prn-example/PRN42/b42-1668-a</d:XAddrs>
        <d:MetadataVersion>75965</d:MetadataVersion>
      </d:ProbeMatch>
    </d:ProbeMatches>
  </s:Body>
</s:Envelope>
```

3 Service Activation and Messaging

3.1 SOAP (Simple Object Activation Protocol)

SOAP, originally defined as Simple Object Access Protocol, is a protocol specification for exchanging structured information in the implementation of web services in computer networks. It relies on XML Information Set for its message format, and usually relies on other application layer protocols, most notably Hypertext Transfer Protocol (HTTP) or Simple Mail Transfer Protocol (SMTP), for message negotiation and transmission.

At this point, REST is favored over SOAP in web programming. Both uses HTTP for transport but REST is more or less AllJoyn CRUD of attribute while SOAP is more like general method invocation. – Google-search "REST vs SOAP" for details..

```
POST /InStock HTTP/1.1
Host: www.example.org
Content-Type: application/soap+xml; charset=utf-8
Content-Length: 299
SOAPAction: "http://www.w3.org/2003/05/soap-envelope"

<?xml version="1.0"?>
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
  <soap:Header>
```

```
  </soap:Header>
  <soap:Body>
    <m:GetStockPrice xmlns:m="http://www.example.org/stock">
      <m:StockName>IBM</m:StockName>
    </m:GetStockPrice>
  </soap:Body>
</soap:Envelope>
```

4 Service Orchestration

4.1 WS-BPEL (Business Process Execution Language)

4.1.1 History

- IBM had "programming in the large" language **WSFL** (Web Services Flow Language)
- Microsoft had its own "programming in the large" language **XLANG**
  - **programming in the large**[1]: e.g. module interconnection language, Meadowview
- **BPML** (Business Process Modeling Language) was proposed popular
- Later, **BPEL** (a.k.a. WS-BPEL, BPEL4WS) became the de facto standard.

4.1.2 Orchestration vs Choreography

- BPEL is an **orchestration** language, not a **choreography** language.
- **orchestration**: executable process that involves message exchanges with other systems, such that the message exchange sequences (i.e. **"protocol"**) are controlled by the orchestration designer
- **choreography**: specifies a protocol for peer-to-per interactions, defining, e.g. the legal sequences of messages exchanged with the purpose of guaranteeing interoperability
  - more of a "specification" than "programming"
  - consider CCS, CSP, π-calculus
  - **NOT executable**
  - ONE GOAL of MEADOWVIEW: Make an *executable* choreography language

4.1.3 Example purchaseOrderProcess

The following are "un-XML-ized" version.

[1] "By large programs we mean systems consisting of many small programs (modules), possibly written by different people. We need languages for programming-in-the-small, i.e. languages not unlike the common programming languages of today, for writing modules. We also need a *module interconnection language* for knitting those modules together into an integrated whole and for providing an overview that formally records the intent of the programmer(s) and that can be checked for consistency by a compiler. We explore the software reliability aspects of such an interconnection language. Emphasis is placed on facilities for information hiding and for defining layers of virtual machines"

```
// IDL -- WSDL definition
message POMessage {
  customerInfoType customerInfo;
  purchaseOrderType purchaseOrder;
};
message InvMessage {
  InvoiceType IVC;
};
message orderFaultType {
  OrderFault problemInfo;
};

message shippingRequestMessage {
  customerInfo info;
};
message shippingInfoMessage {
  shippingInfo info;
};
message scheduleMessage {
  scheduleInfo schedule
};

// Ports: supported by purchase order process
// non-first-class version of SV interface
port purchaseOrderPort {
  operation sendPurchseOrder(input POMessage po;
                             output InvMessage inv;
                             fault orderFaultType failure);
};
port invoiceCallbackPort {
  operation sendInvoice(input InvMessage inv);
};
port shippingCallbackPort {
  operation sendSchedule(input scheduleMessage msg);
};

// Ports: supported by shipping services
port shippingPort {
  operation requestShipping(input shippingRequestMessage msg,
                            output shippingInfoMessage msg,
                            fault orderFaultType failure);
};

// Ports: supported by invoice services
port computePricePort {
  operation initiatePriceCalculation(input POMessage msg);
  operation sendShippingPrice(input shippingInfoMessage msg);
};

// Ports: supported by production scheduling process
port schedulingPort {
```

```
  operation requestProductionScheduling(input POMessage msg);
  operation sendShippingSchedule(input scheduleMessage msg);
};
```

Let P_0 and P_1 be processes, where P_i already defines its services using WSDL (IDL). "Links" are used to model relationships between partner processes – e.g. between P_0 and P_1.

```
// Link -- Partner Links
link purchasingLink {
  // role <role-type> <role-name>
  role purchaseOrderPort purchaseService;
};
link invoicingLink {
  role computePricePort invoiceService;
};
link shippingLink {
  role shippingPort shippingService;
  role shippingCallbackPort shippingRequester;
};
link schedulingLink {
  role schedulingPort schedulingService;
};

// PROCESS
process purchaseOrderProcess {
  // partner links
  puchasingLink purchasing(purchasingService /*myrole*/);
  invoicingLink invoicing(invoiceRequester /*myrole*/,
                          invoiceService /*partnerrole*/);
  shippingLink shipping(shippingRequester /*myrole*/,
                        shippingSerivce /*partnerrole*/);
  schedulingLink scheduling(schedulingService /*partnerrole*/);

  // variables
  POMessage PO;
  InvMessage Invoice;
  shippingRequestMessage shippingRequest;
  shippingInfoMessage shippingInfo;
  scheduleMessage shippingSchedule;

  // fault handlers
  faultHandler {
    catch (cannotCompleteOrder(POFaultType POFault)) {
      reply(purchasing /*link*/,
    };
  };

  sequence {
    receive(purchasing /*link - shippingPort*/,
            sendPurchaseOrder /*operation*/, PO /*bound*/);
```

```
    // arrows
    arrow ship-to-invoice;
    arrow ship-to-scheduling;

    // sequence #1: CENTER
    sequence {
      $shippingRequest.customerInfo = $PO.customerInfo;

      // decide on shipper
      invoke(shipping /*link - shippingPort*/,
             requestShipping /*operation*/,
             shippingRequest /*input*/,
             shippingInfo /*output */,
             source=ship-to-invoice /* arrow */);

      // arrange logistics
      receive(shipping /*link - shippingCallbackPort*/,
              sendSchedule /*operation*/, shippingSchedule /*bound*/,
              source=ship-to-scheduling /*arrow*/);
    };

    // sequence #2: LEFT (INVOICE)
    sequence {
      // initial price calculation
      invoke(invoicing /*link - computePricePort */,
             initiatePriceCalculation /*operation*/,
             PO /*input*/);

      // complete price calculation
      invoke(invoicing /*link - computePricePort */,
             sendShippingPrice /*operation*/,
             shippingInfo /*input*/,
             target=ship-to-invoice /*arrow*/);

      receive(invoicing /*link - invoiceCallbackPort*/,
              sendInvoice /*operation*/, Invoice);
    };

    // sequence #3: RIGHT (SHIPPING)
    sequence {
       ...
    };
  };
};
```

5 Service Choreography

5.1 WS-CDL (Web Services Choreography Description Language)

5.1.1 WS-CDL model overview

WS-CDL describes interoperable, peer-to-peer collaborations between participants. In order to facilitate these collaborations, services commit to mutual responsibilities by establishing formal relationships. Their collaboration takes place in a jointly agreed set of ordering and constraint rules, whereby information is exchanged between the participants.

The WS-CDL model consists of the following entities:

- **roleType, relationshipType and participantType**: Within a choreography, information is always exchanged between participants within or across trust boundaries. All interactions occur between roles being exhibited by participants, and are constrained by a relationship. Within WS-CDL, a participant is abstractly modeled by a participantType, a role by a roleType, and a relationship by a relationshipType:
  - A **participantType** groups together those parts of the observable behavior that must be implemented by the same logical entity or abstract organization
  - A **roleType** enumerates potential observable behavior a participantType can exhibit in order to interact
  - A **relationshipType** identifies the mutual commitments that must be made for collaborations to be successful
- **informationType, variable and token**: A variable contains information about commonly observable objects in a collaboration, such as the information exchanged or the observable information of the roleTypes involved. A token is an alias that can be used to reference parts of a variable. Information exchange variables, state capturing variables and tokens have informationTypes that define the type of information the variable contains or the token references
- **choreography**: A choreography defines collaborations between interacting participantTypes:
  - **choreography life-line**: The choreography life-line expresses the progression of a collaboration. Initially, the collaboration is established between participants, then work is performed within it and finally it completes either normally or abnormally
  - **choreography exception blocks**: An exception block specifies what additional actions should occur when a choreography behaves in an abnormal way
  - **choreography finalizer blocks**: A finalizer block specifies additional actions that should occur to modify the effect of an earlier successfully completed choreography, for example to confirm or undo the effect
- **channelType**: A channel realizes a point of collaboration between participantTypes by specifying where and how information is exchanged. Within WS-CDL, channels are abstractly modeled as channelTypes
- **workunit**: A workunit prescribes the constraints that must be fulfilled for making progress, and thus performing work, within a choreography
- **activities and ordering structures**: Activities describe the actions performed within a choreography. Ordering structures combine activities with other ordering structures in a nested structure to express the ordering rules of actions performed within a choreography

- **interaction activity**: An interaction is the basic building block of a choreography. It results in an exchange of information between participants and possible synchronization of their observable information changes
- **semantics**: Semantics allow the creation of descriptions that can record the semantic definitions of every component in the model

5.2 WSCI (Web Service Choreography Interface)

5.3 WSCL (Web Service Conversation Language)