Web services composition with WS-BPEL and OWL-S

Riina Maigre

09.02.2006

Outline

Web services overview

- What is a web service?
- Web services architecture

Web services composition

- What it means?
- Composition, orchestration, choreography

WS-BPEL

- About WS-BPEL
- WS-BPEL examples

OWL-S

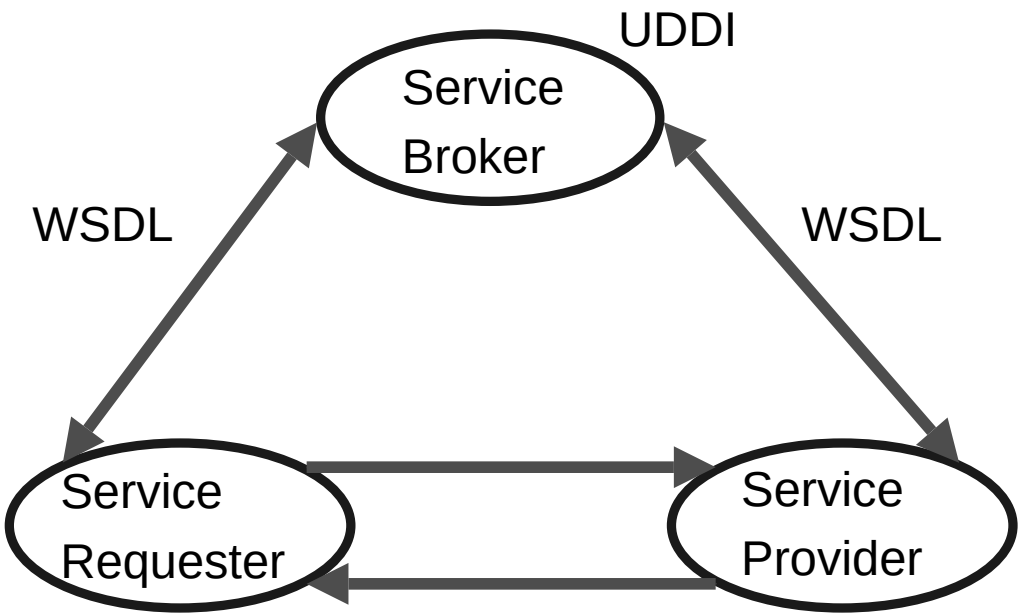
- About OWL-S
- OWL-S examples

Comparision of WS-BPEL and OWL-S

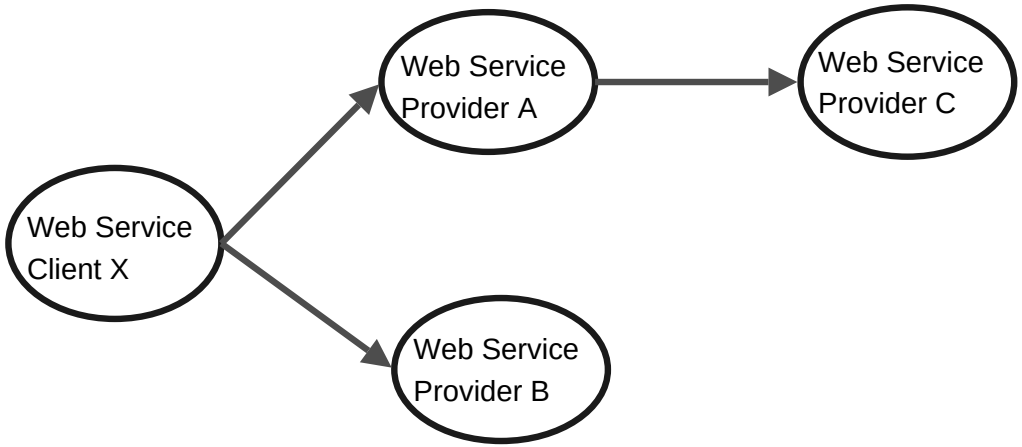
What is a web service?

- Web service is web-based enterprise application that use open, XML-based standards and transport protocols to exchange data with calling clients.
- Web service is a software component that is described via WSDL and is accessible via standard network protocols such as but not limited to SOAP over HTTP.
- Web service should be
  - Based on open standards
  - Platform independent
  - Application independent
  - Enable to share data and resources

Web services architecture



Web services composition



- Is the task of combining and linking existing web services to create new web processes in order to add value to the collection of services.
- Two approaches:
  - Industry solution: WS-BPEL, XLANG, WSFL, WSCI, BPML
  - Semantic web solution: OWL-S, OWL, DAML-S, DAML+OIL

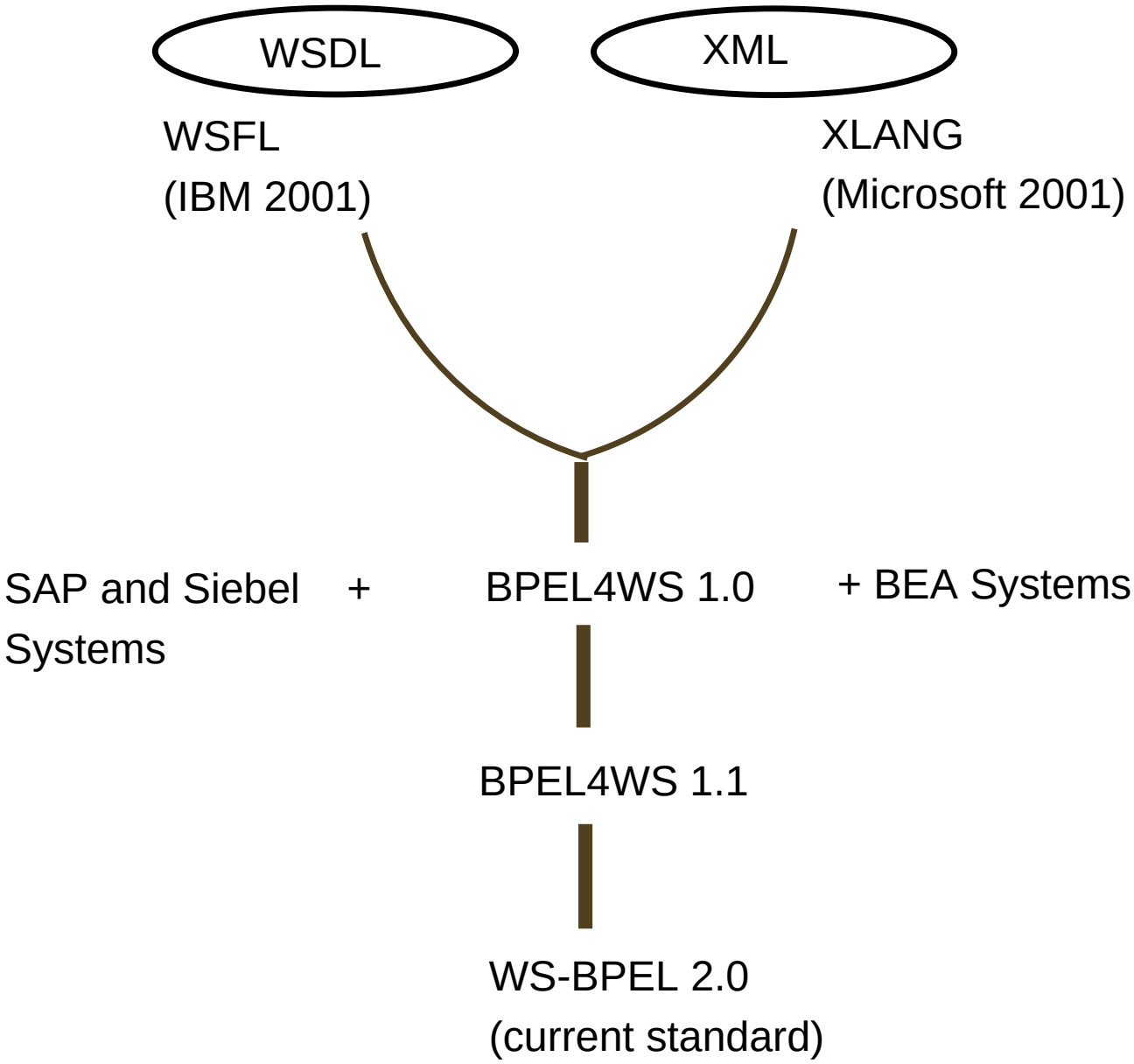
Composition: orchestration and choreography

- Private processes manage services inside a given organization – WS orchestration
- Public processes manage services across several organizations – WS choreography

WS-BPEL

- WS-BPEL stands for Web Services Business Process Execution Language.
- WS-BPEL is XML grammar defining and standardizing structures necessary for web services orchestration.
- Composition is based on pre-modeled workflow.
- In WS-BPEL everything is a service.

WS-BPEL family tree



A little about WS-BPEL terminology

- Activities – message exchange or intermediate result transformation
- Process – the composition result
  - A process consists of a set of activities.

Activities in WS-BPEL

- From BPEL4WS
  - Basic activities – the ones that do something.
    - invoke
    - receive
    - reply
    - wait
    - empty
  - Structured activities – the ones that organize basic activities without doing anything by themselves.
    - sequence
    - flow
    - switch
    - while
    - pick
- Newer additions
  - If
  - Repeat until
  - ForEach
  - Exit

Activities in WS-BPEL

- From BPEL4WS
  - Basic activities – the ones that do something.
    - invoke
    - receive
    - reply
    - wait
    - empty
  - Structured activities – the ones that organize basic activities without doing anything by themselves.
    - sequence
    - flow
    - switch
    - while
    - pick
- Newer additions
  - If
  - Repeat until
  - ForEach
  - Exit

Activities in WS-BPEL

- From BPEL4WS
  - Basic activities – the ones that do something.
    - invoke
    - receive
    - reply
    - wait
    - empty
  - Structured activities – the ones that organize basic activities without doing anything by themselves.
    - sequence
    - flow
    - switch
    - while
    - pick
- Newer additions
  - If
  - Repeat until
  - ForEach
  - Exit

Other control flow features in WS-BPEL

- Variables – to hold messages or data
- Error handling – allows to catch and handle errors
- Compensation handling – allows to undo steps that have already been completed
- Message correlation – allows processes to participate in stateful conversations

WS-BPEL example – process

```
<process name ="TripHandling"... >
  <partnerLinks>
    <partnerLink name ="Customer"
                 myRole ="TripHandlingAgent"
      partnerLinkType ="ExternalServiceLink"
          partnerRole ="CustomerAgent"/>
    <partnerLink name ="FlightService"
                 myRole ="TripHandlingAgent"
      partnerLinkType ="InternalServiceLink"
          partnerRole ="FlightServiceAgent"/>
    <partnerLink name ="HotelService"
                 myRole ="tripHandlingAgent"
      partnerLinkType ="InternalServiceLink"
          partnerRole ="HotelServiceAgent"/>
  </partnerLinks>
  <variables>  ...  </variables>
  ...
</process>
```

WS-BPEL example – variables

```
<variables>
  <variable name ="OrderEvent"
     messageType ="OrderEventType"/>
  <variable name ="TripRequest"
     messageType ="TripRequestType"/>
  <variable name ="FlightRequest"
     messageType ="FlightRequestType"/>
  <variable name ="HotelRequest"
     messageType ="HotelRequestType"/>
  <variable name ="BookingFailure"
     messageType ="BookingFailureType"/>
</variables>
```

WS-BPEL example – activities 1

```
<sequence>
  <receive partnerLink ="Customer"
                portType ="pt1"
               operation ="CToCI"
                variable ="OrderEvent"
  </receive>
  <flow>
    <invoke partnerLink ="HotelService"
                 portType ="pt2"
                operation ="CIToHS"
              inputVariable ="HotelRequest">
    </invoke>
    ...
  </flow>
</sequence>
```

WS-BPEL example – activities 2

```
<flow>
    <receive partnerLink ="HotelService"
                  portType ="pt4"
                 operation ="HSToEVAL1"
                  variable ="HotelRequest">
          <correlations>
             <correlation set="tripIdentification"/>
          </correlations>
   </receive>
   <receive partnerLink ="FlightService"
                ...
   </receive>
</flow>
```

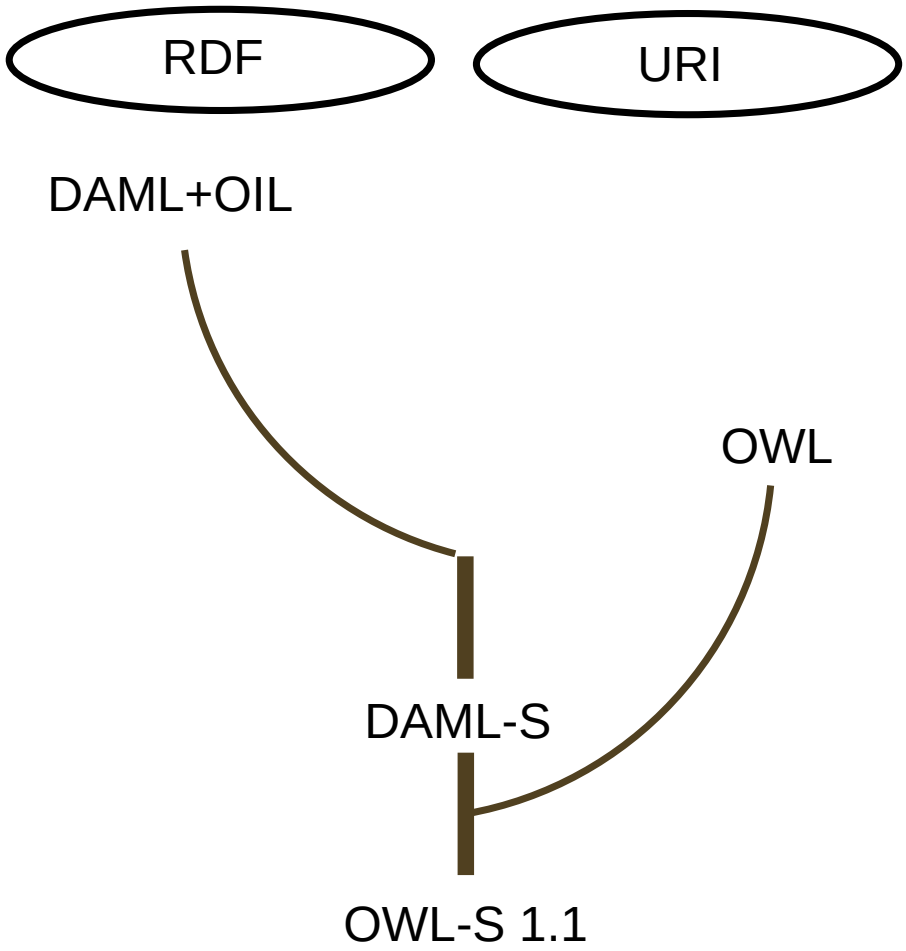
WS-BPEL example – activities 3

```
<switch>
  <case condition ="condition1">
    <invoke partnerLink ="Customer"...> </invoke>
  </case>
  <otherwise>
    <flow>
      <invoke partnerLink ="Customer"...> </invoke>
      <switch>
        <case condition="condition2">
          <invoke partnerLink ="HotelService"...>
          </invoke>
        </case>
        <otherwise>
          <invoke partnerLink ="FlightService"...>
          </invoke>
      </switch>
      ...
</switch>
```

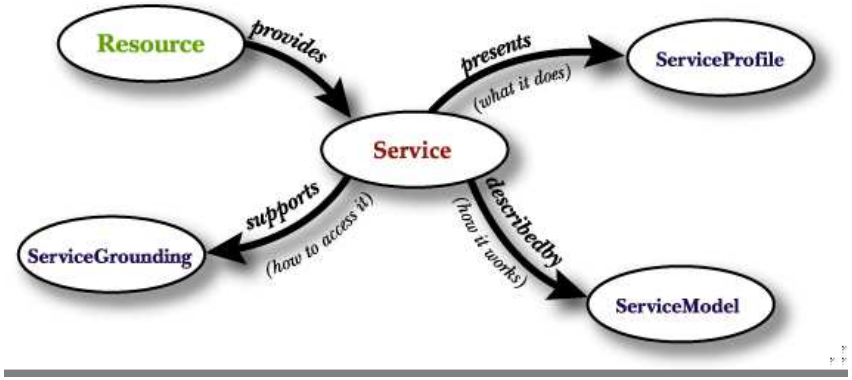
About OWL-S

- OWL-S stands for Web Ontology Language for Services (or Ontology Web Language for Services)
- It is an ontology that enables automatic service discovery, invocation, composition and execution monitoring.
- Composition is based on pre- and post-conditions.

OWL-S family tree



Full OWL-S



- ServiceProfile – high-level description of the service
- ServiceModel – detailed description of the service in which it is modeled as a process
- ServiceGrounding – binding level information of how a client can access the service

OWL-S composition templates

- Sequence
- Split
- Unordered
- Split + Join
- Choice
- Condition
- If-Then-Else
- Iterate
- Repeat-Until

OWL-S example 1

```
<process:CompositeProcess rdf:ID="BravoAir_Process">
 <rdfs:label>This is the top level process for BravoAir
                                                </rdfs:label>
  <process:composedOf>
    <process:Sequence>
      <process:components rdf:parseType="Collection">
        <process:AtomicProcess rdf:about=
                              "#GetDesiredFlightDetails"/>
        <process:AtomicProcess rdf:about=
                              "#SelectAvailableFlight"/>
        <process:CompositeProcess rdf:about=
                              "#BookFlight"/>
        </process:components>
    </process:Sequence>
  </process:composedOf>
</process:CompositeProcess>
```

OWL-S example 2

```
<process:CompositeProcess rdf:ID="BookFlight">
  <process:composedOf>
    <process:Sequence>
      <process:components rdf:parseType="Collection">
        <process:AtomicProcess rdf:about="#Login"/>
        <process:AtomicProcess rdf:about=
                                  "#ConfirmReservation"/>
      </process:components>
    </process:Sequence>
  </process:composedOf>
</process:CompositeProcess>
```

OWL-S example 3

```
<process:AtomicProcess rdf:ID="LogIn">
  <process:hasInput rdf:resource="#AcctName_In"/>
  <process:hasInput rdf:resource="#Password_In"/>
</process:AtomicProcess>

<process:Input rdf:ID="AcctName_In">
  <process:parameterType rdf:resource=
                          "&concepts;#AcctName">
</process:Input>
<process:Input rdf:ID="Password_In">
  <process:parameterType rdf:resource=
                          "&concepts;#Password">
</process:Input>
```

Comparison of WS-BPEL and OWL-S

WS-BPEL	OWL-S
Industry driven	Academy driven
Lacks semantics described in WSDL	Majority of services are
Runtime engine	Planner
Has good control over workflow at design time. Not too flexible at runtime	More flexible at runtime
Inputs and outputs of the service described in WSDL	Pre- and post-conditions
Set of choices is pre-determined	Choises are based on goals

For further reading

- Matthieu Riou

  WS-BPEL Guide 2004

- Oasis WS-BPEL Technical committee

  Web Services Business Process Execution Language Version 2.0 January 2006

- Michael Hu

  Web Services Composition, Partition, and Quality of Service in Distributed System Integration and Re-engineering 2004

- Biplav Srivastava, Jana Koehler

  Web Service Composition - Current Solutions and Open Problems 2003

- http://www.ebpml.org/

- http://java.sun.com/webservices/

- http://www.daml.org/services/owl-s/1.1/

WS-BPEL example – process

```
<process name ="TripHandling">
  <partners>
    <partner name ="Customer"
            myRole ="TripHandlingAgent"
  serviceLinkType ="ExternalServiceLink"
      partnerRole ="CustomerAgent"/>
    <partner name ="FlightService"
            myRole ="TripHandlingAgent"
  serviceLinkType ="InternalServiceLink"
      partnerRole ="FlightServiceAgent"/>
    <partner name ="HotelService"
            myRole ="tripHandlingAgent"
  serviceLinkType ="InternalServiceLink"
      partnerRole ="HotelServiceAgent"/>
  </partners>
  <containers>  ...  <containers>
  ...
</process>
```

WS-BPEL example – containers

```
<containers>
  <container name ="OrderEvent"
      messageType ="OrderEventType"/>
  <container name ="TripRequest"
      messageType ="TripRequestType"/>
  <container name ="FlightRequest"
      messageType ="FlightRequestType"/>
  <container name ="HotelRequest"
      messageType ="HotelRequestType"/>
  <container name ="BookingFailure"
      messageType ="BookingFailureType"/>
</containers>
```

WS-BPEL example – activities 1

```
<sequence>
  <receive partner="Customer"
           portType ="pt1"
          operation ="CToCI"
          container ="OrderEvent">
  </receive>
  <flow>
    <invoke partner  ="HotelService"
            portType ="pt2"
           operation ="CIToHS"
     inputContainer  ="HotelRequest">
    </invoke>
    ...
  </flow>
</sequence>
```

WS-BPEL example – activities 2

```
<flow>
    <receive partner ="HotelService"
             portType ="pt4"
            operation ="HSToEVAL1"
            container ="HotelRequest">
   </receive>
   <receive partner ="FlightService"
            portType ="pt5"
           operation ="FSToEVAL1"
           container ="FlightRequest">
   </receive>
</flow>
```

WS-BPEL example – activities 3

```
<switch>
  <case condition ="condition1">
    <invoke partner ="Customer"...> </invoke>
  </case>
  <otherwise>
    <flow>
      <invoke partner ="Customer"...> </invoke>
      <switch>
         <case condition="condition2">
           <invoke partner ="HotelService"...>
           </invoke>
         </case>
         <otherwise>
           <invoke partner ="FlightService"...>
           </invoke>
        </otherwise>
      ...
</switch>
```