

# Semantic Web

## Web Services and Semantic Web Services

F. Abel, N. Henze, D. Krause

IVS Semantic Web Group

15.01.2009

## 1 Web Services

- *Services* on the Web
- Service Oriented Architectures
- Standards for Web Services
  - SOAP
  - WSDL
  - UDDI - Universal Description, Discovery and Integration
- *Semantic* Web Services
  - Why do we need Semantic Web Services?
  - Semantic Web Services – Challenges
  - OWL-S
  - Semantic Web Services Architecture

traditional approach: access to data via “browser” (Netzkappe)

Web Service: is a piece of software which

- 1 is available via a server
- 2 provides a certain functionality as a blackbox
- 3 is accessible via standard internet protocols (SOAP)
- 4 has well-defined interface descriptions (WSDL)

## Example (Google)

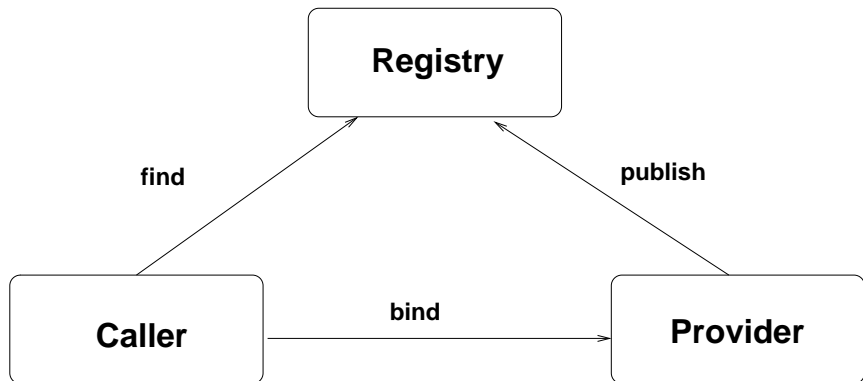
- *“Google without browser”*
- *Google's Web **Services**: Search, Spell-Checking*
  - *application calls Google: Search request as SOAP message*
  - *Google answers application: Result as SOAP message*
- *“With Google Web-APIs, your computer can do the searching for you”*
- *<http://www.google.de/apis/>*

## Definition

Web Services 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 request to complicated business processes. (IBM Web Services tutorial)

## Benefits of Web Services

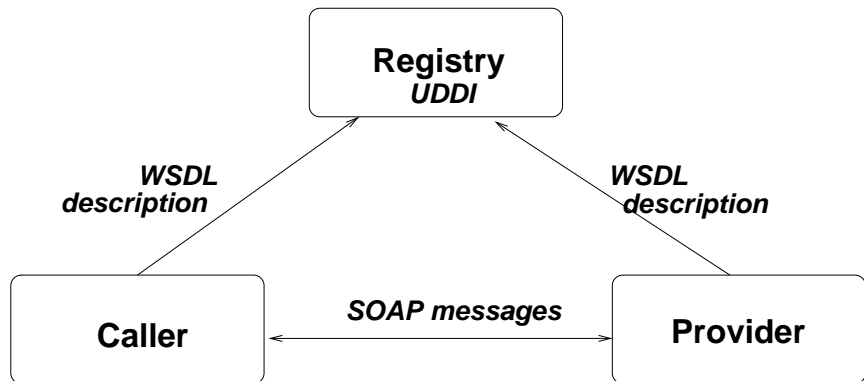
- Web service technology makes functionality available independently of many aspects of the actual implementation of the Web Service
- A Web Service can be located anywhere in the network
- Web Service requesters can be again Web services (interoperability, composition)
- Web Services are seen as a promising new technology for machine-to-machine interactions across application, enterprise, and web boundaries
- particularly for e-Commerce:
  - provide installation-free business logic access
  - provide installation-free application system access



**publish** a description of a service in a registry

**find** a description of a service

**bind** a service for actual use, using the service's description

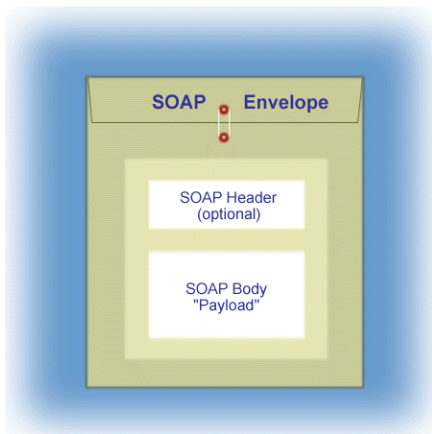


**UDDI** Universal Description, Discovery and Integration

**WSDL** Web Service Description Language

**SOAP** Simple Object Access Protocol

- SOAP: XML-based protocol for exchanging information in a distributed environment.
- SOAP separates message content (body) and additional information (header)
- A SOAP message consists of a mandatory **Envelope**, an optional **Header**, and a mandatory **Body**



## Envelope

- the Envelope element is the top element of the XML document
- can contain namespace declarations as well as additional attributes such as an encoding style
- an encoding style identifies the data types recognized by SOAP messages, and specifies rules how these data types are serialized, that is, transformed for shipped over the Web.

## Header

- The header is represented by a Header element
- Optional
- If a header is included in a SOAP message, it must be the first child of the Envelope element
- The header, through attributes, extends the SOAP message, e.g. authentication, payment, communication control, etc.

## Body

- The body, represented by a Body element, contains the information intended for the final destination (contains the actual SOAP message)
- The Body element can optionally contain a Fault element that is used to hold error and status information returned by a processing node
- The Body element must be an immediate child element of a SOAP Envelope element; if the envelope contains a header, the Body element must immediately follow the Header element.
- The body can in general contain arbitrary XML elements; the structure of the body is application dependent and can be specified via a namespace or WSDL

## Example (A SOAP message)

*In this example, a GetStockPrice request is sent to a service. The request has a StockName parameter, and a Price parameter will be returned in the response.*

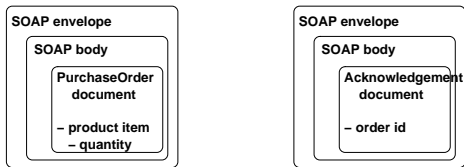
### **The SOAP request:**

```
<soap:Envelope xmlns:soap="http://www.w3.org/2001/12/soap-envelope">
  <soap:Body xmlns:m="http://www.stock.org/stock">
    <m:GetStockPrice>
      <m:StockName>IBM</m:StockName>
    </m:GetStockPrice>
  </soap:Body>
</soap:Envelope>
```

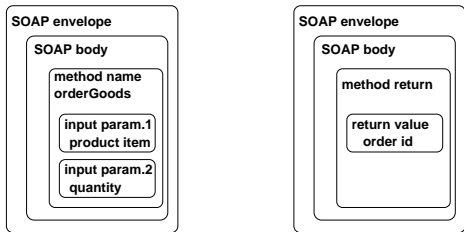
### **And the SOAP response:**

```
<soap:Envelope xmlns:soap="http://www.w3.org/2001/12/soap-envelope">
  <soap:Body xmlns:m="http://www.stock.org/stock">
    <m:GetStockPriceResponse>
      <m:Price>34.5</m:Price>
    </m:GetStockPriceResponse>
  </soap:Body>
</soap:Envelope>
```

# SOAP interaction styles



(a) Document-style interaction



(b) RPC-style interaction  
[RPC: Remote Procedure Call]

**Nota bene:** SOAP does not impose any particular interaction style!

- specification of protocol which is used for exchanging the SOAP messages
- currently: SOAP messages are typically binded to HTTP/HTTPS
- other bindings like SMTP are possible
- **Nota bene:** SOAP does not impose any transport protocol!

# The SOAP HTTP Binding:

- A SOAP request could be an HTTP POST or an HTTP GET request.
- The HTTP POST request specifies at least two HTTP headers: Content-Type and Content-Length.

**Content-Type** The Content-Type header for a SOAP request and response defines the MIME type for the message and the character encoding (optional) used for the XML body.

**Example:**

```
POST /item HTTP/1.1
Content-Type: application/soap+xml; charset=utf-8
```

**Content-Length** The Content-Length header for a SOAP request and response specifies the number of bytes in the body.

**Example:**

```
POST /item HTTP/1.1
Content-Type: application/soap+xml; charset=utf-8
Content-Length: 250
```

## Example (Sending a GetStockPrice request to a server)

```
POST /InStock HTTP/1.1
```

```
Host: www.stock.org
```

```
Content-Type: application/soap+xml; charset=utf-8
```

```
Content-Length: nnn
```

```
<?xml version="1.0"?>
```

```
<env:Envelope xmlns:env="http://www.w3.org/2002/06/soap-envelope">
```

```
<env:Header>
```

```
<t:transactionID
```

```
xmlns:m="http://www.stock.org/stock"
```

```
env:role="http://www.w3.org/2002/06/soap-envelope/role/next"
```

```
env:mustUnderstand="true">
```

```
7411
```

```
</t:transactionID>
```

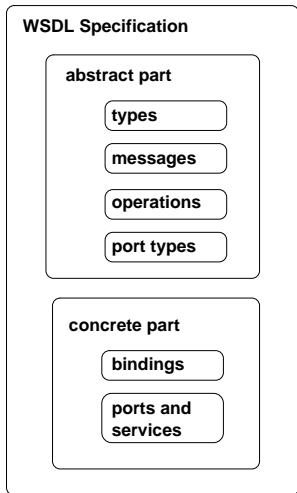
```
</env:Header>
```

## Example (Sending a GetStockPrice request to a server)

```
<env:Body>
  <m:orderGoods xmlns:m="http://www.stock.org/stock"
    env:encodingStyle="http://www.w3.org/2002/06/soap-encoding"
    <m:productItem>
      <name> ACME Softener </name>
    </m:productItem>

    <m:quantity>
      35
    </m:quantity>
  </m:orderGoods>
</env:Body>
</env:Envelope>
```

- Web service expose a software-oriented view of a business or consumer function with which applications may interact over the network
- To enable such interaction, a Web service must be described and advertised to its potential users:
  - the operations it provides
  - the data it expects to receive (input)
  - the results it delivers (output)
  - what communication protocol or transport it supports (binding)
- WSDL (Web Services Description Language) is an XML-based language for describing Web services and how to access them
- Typically, WSDL is used with SOAP, and the WSDL specification includes a SOAP binding
- WSDL 2.0: Working Draft (May 2005)
- WSDL 2.0: Recommendation (June 2007)



**<types>:** the data types used by the Web service

**<messages>:** the messages used by the Web service

**<operations>:** the operations performed by the Web service

**<portType>:** defines the Web service (the operations that it can perform, and the messages involved)

**<binding>:** defines the message encoding and protocol details for a port type

**<port>:** specifies a network address for a binding where the implementation of a port type can be found

**<service>:** a logical grouping of ports

**Nota Bene:** WSDL doesn't say anything about what Web services can do, nor how they do it!

- WSDL needs a type system so that the data being exchanged can be correctly interpreted
- default type: XML Schema
- **First step** in defining a WSDL interface is to identify and define all the data structures that will be exchanged as parts of messages

## Example (WSDL - Types)

```
<types>
  <schema ...>
    <complexType name = "PurchaseOrder">
      <element name="NameofProduct" type="xsd:string"/>
      <element name="Price" type="xsd:integer"/>
    </complexType>
    ...
  </schema>
</types>
```

- **Second step** in defining a WSDL interface is to define the messages which can be exchanged.
- Messages consist of one or more parts
- Each part is characterized by name and type

## Example (WSDL - Messages)

*A WSDL message with one part:*

```
<message name="PurchaseOrderRequest">  
  <part name="MyChristmasOrder" type="mynamespace:PurchaseOrder"/>  
</message>
```

*A WSDL message with two parts:*

```
<message name="doGoogleSearch">  
  <part name="param1" type="mynamespace:param1"/>  
  <part name="param2" type="mynamespace:param2"/>  
</message>
```

- **Third step** in defining a WSDL interface is to define the actions that can be performed on the data
- Operations are defined so that the Web services now know how to interpret the data and what, if any, data is to be returned on reply
- Types of operations:
  - **One-Way:** One single message; the client invokes a service by sending a message
  - **Request-response:** Exchange of two messages; the service is invoked and a response is sent
  - **Solicit-response:** Exchange of two messages; the service makes a request and expects a response
  - **Notification:** One single message; the service sends a message
- Synchronous operations: use request-response or solicit-response operations
- Asynchronous operations: use one-way or notification operations
- optionally, operations can have a fault message
- it is not possible to relate two output messages in one single operation

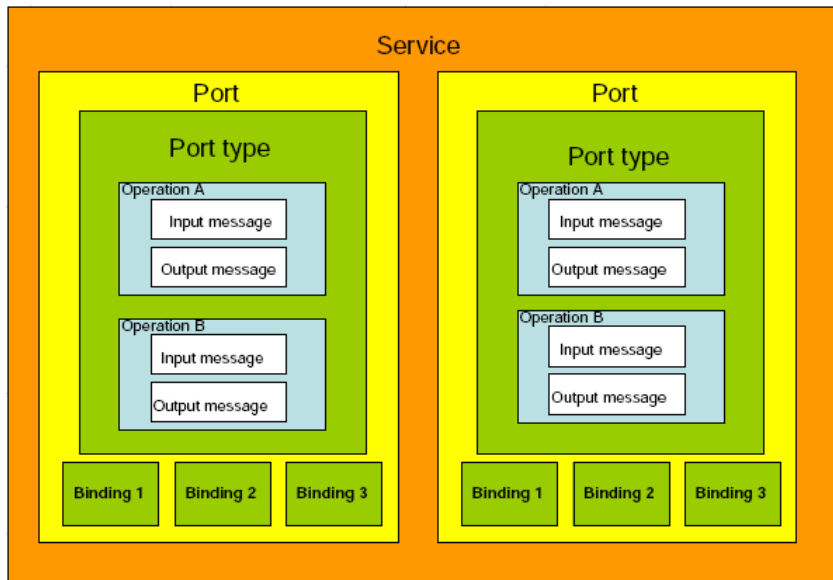
## Example (WSDL - Operations)

```
<operation name="ProcessPurchaseOrder">  
  <input message="PurchaseOrderRequest"/>  
  <output message="PurchaseOrderResult"/>  
</operation>
```

- **Fourth step** in defining a WSDL interface is to logically group operations into so-called port types
- WSDL 1.2 allows a port type to be extended from other port types

## Example (WSDL - Port Types)

```
<portType name="PurchaseOrderPortType">  
  <operation name="ProcessPurchaseOrder">  
    <input message="PurchaseOrderRequest"/>  
    <output message="PurchaseOrderResult"/>  
  </operation>  
  <operation name="CancelPurchaseOrder">  
    <input message="CancelPurchaseOrderRequest"/>  
    <output message="CancelPurchaseOrderResult"/>  
  </operation>  
</portType>
```



- A binding specifies the message encoding and protocol bindings for all operations and messages defined in a given port type

protocol binding:

- e.g. a binding could specify that the messages of an operation have to be communicated using the SOAP protocol and HTTP transport bindings
- **Nota bene:** other protocols than SOAP can be used as well!

encoding:

- two possible encodings: literal and SOAP
- **literal** encoding takes the WSDL types defined in XML Schemas and *"literally"* uses those definitions to represent the XML content of messages; the abstract WSDL types become concrete types!
- **SOAP encoding** takes the XML schema definitions as abstract entities, and translates them into XML using SOAP encoding rules defined as part of SOAP 1.2

## Ports:

- also known as *EndPoints*
- combine the binding information with a network address (specified by a URI) at which the implementation of the port type can be accessed

## Service:

- logical grouping of ports
- can combine very different port types; in practice, however it is likely that a WSDL service will group related ports

- UDDI provides a service registry to locate web services
- A UDDI can be thought of as a DNS for business applications
- Classification of businesses and services according to standard taxonomies

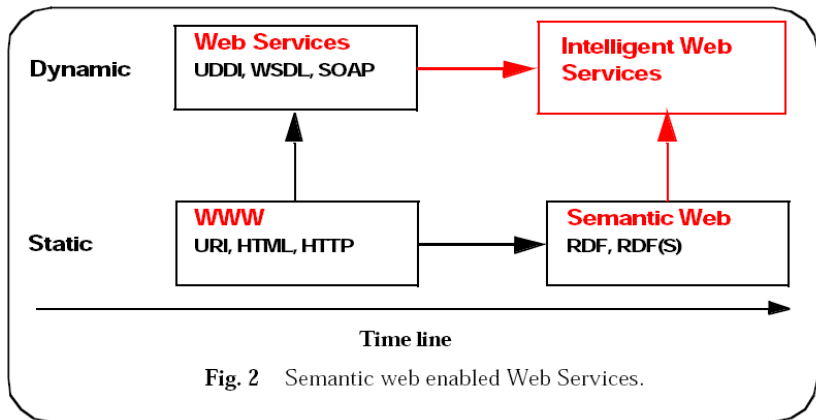
Two main goals for service discovery:

- support developers in finding information about services, so that they know how to write clients that interact with those services
- enable dynamic binding, by allowing clients to query the registry and obtain references to services of interest

- simple way of categorizing the information contained in the UDDI:  
describe what each type of information is used for
- analogy with the telephone directory:
  - White pages:
    - listings of organization (contact information)
    - and the services these organizations provide
    - *Using the registry as a white pages catalogue, UDDI clients can find Web services provided by a given business*
  - Yellow pages:
    - classifications of both companies and web services according to taxonomies
    - these taxonomies can be standardized or use-defined
    - *Through yellow pages, it is possible to search for services based on the category they belong to, according to a given classification scheme*

## Green pages:

- describes how a given Web service can be invoked
- it is provided by means of pointers to service description documents, typically stored outside the registry
- *Through green pages, the Web service descriptions can be accessed*



Source: "The Web Service Modeling Framework WSMF". D. Fensel and C. Bussler; <http://www.wsmo.org/papers/publications/wsmf.paper.pdf>

# Why do we need Semantic Web Services?

- UDDI, WSDL and SOAP facilitate the means to advertise, describe, and invoke Web services
- UDDI, WSDL and SOAP do not say anything about **what services can do** nor **how they do it** in a machine understandable and processable way
- lack of proper support for semantics: the main obstacle for automated *discovery, combination, and execution* of Web services
- Goal must be: minimize human intervention; assembly of Web Services should be done in a task-driven, automatic way.

*Semantic Web Services: combining Web services technology with Semantic Web technology.*

*“Semantic differences remain the primary roadblock to smooth application integration, one which Web Services alone won't over-come. Until someone finds a way for applications to understand each other, the effect of Web services technology will be fairly limited. When I pass customer data across [the Web] in a certain format using a Web Services interface, the receiving program has to know what that format is. You have to agree on what the business objects look like. And no one has come up with a feasible way to work that out yet - not Oracle, and not its competitors...”*

Oracle Chairman and CEO Larry Ellison

## Functional requirements on Web Services

**Publishing:** how it is published / advertised

**Discovery:** how can other applications find this service

*Example: A service requester might say: “Locate all the services that can solve mathematical equations”. An automatic service discovery engine will then surf all available repositories in search of services that fulfill the given task. The list of available services will probably be huge, so the user might impose some limitations, functional – how the service is provided – and non-functional – execution time, cost, and so on – in order to get an accurate and precise set of services*

**Selection:** how applications can decide whether to use this service

*Example: After discovery of two fitting service one has to decide which one to choose. One of them has very strong security requirements, it is expensive and really reliable, while the other one is not so reliable, but is cheaper and is provided by our favorite software vendor. All these requirements might have been previously provided by the user, even before the discovery phase, allowing carrying this task in a more or less automatic way. Also the user could browse the whole list of selected services and select the one that interests him/her most.*

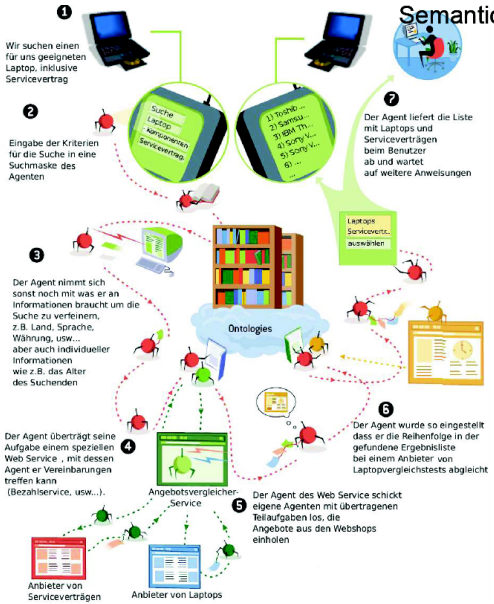
*Ranking of the results is important!*

**Composition:** whether and how this service accesses other services  
*A service requester might say: “Compose available services to solve the following set of mathematical operations  $[(a*b)+c)-d]$ ”. during discovery, different multiplication, addition, and subtraction services might have been found, each one of them with its particular functional and non-functional attributes. Let us suppose that some multiplication services have been located, but they are all too slow and expensive, so we are not interested in using them. Instead we want to use additions to perform the multiplication. Such knowledge, the fact that multiple addition can realize multiplication, should be stated in some domain knowledge in a way that the service composer can understand to present all different possibilities to solve our set of operations.*

**Invocation:** how applications can invoke this service

**Deployment:** how this service’s evolution is managed

# Semantic Web Services



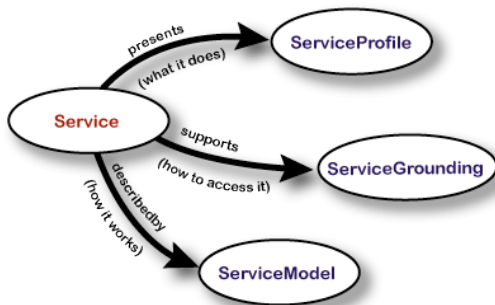
## OWL-S<sup>1</sup>:

- OWL-S is a W3C Member Submission (*submitted in November 2004, not standardized yet*):  
<http://www.w3.org/Submission/OWL-S/>
- OWL-S: Semantic Markup for Web Services
- OWL-S is an ontology that allows to describe Web Services semantically so that the *Challenges* mentioned above can be realized.
- Main parts of the OWL-S ontology: service profile, process model, and grounding

---

<sup>1</sup>The following slides on OWL-S are based on  
<http://www.w3.org/Submission/OWL-S/>

# Structure of the OWL-S ontology



**service profile** The service profile (*ServiceProfile*) tells “what the service does”.

**process model** The service model (*ServiceModel*) tells a client “how to use the service”.

**grounding** The grounding (*ServiceGrounding*) specifies the details of “how to access the service”.

- the service profile provides a way to describe (1) the services offered by the providers, and (2) the services needed by the requesters.
- Challenges tackled by the service profile: *Publishing* (advertisement), *Discovery*, and *Selection*.
- once a service has been selected the profile is useless
- issues described in a service profile:
  - what is accomplished by the service
  - limitations on service applicability
  - quality of service
  - requirements that the service requester must satisfy to use the service successfully

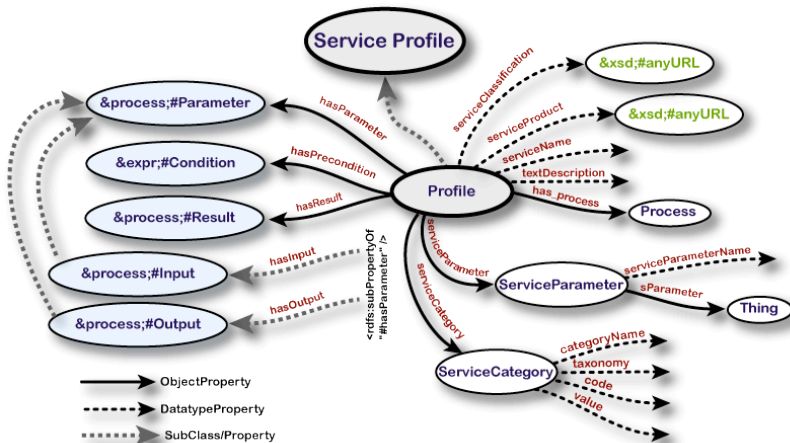


Figure: Extract of the Profile part of the OWL-S ontology.

## Selected properties of the service profile:

**serviceName**, **textDescription**, **contactInformation** human-readable information

**hasInput**, **hasOutput** instances of Inputs/Outputs as defined in the process model

**hasPrecondition** specifies one of the preconditions of the service and ranges over a Precondition. The Web Service process cannot be performed successfully unless the precondition (e.g. an expression ) is true. Example:

```
<hasPrecondition> <expr:KIF-Expression>
  <expr:expressionBody>
    (!agent:knownValue (!ecom:credit_card_num ?cc) ?num)
  </expr:expressionBody>
</expr:KIF-Expression>
</hasPrecondition>
```

**hasResult** specifies one of the results of the service. Furthermore, it specifies under what conditions the outputs are generated.

- Challenges tackled by the process model: *Selection* (e.g. preconditions) and *Composition*.
- Web Services are viewed as a *process*
- Purposes of a process:
  - 1 return some new information based on information it is given and the world state ( $\rightarrow$  *inputs/outputs of the process*)
  - 2 produce a change in the world ( $\rightarrow$  *preconditions and effects of the process*)
- issues described in a process model:
  - the semantic content of requests
  - the conditions under which particular outcomes will occur (where necessary, the step by step processes leading to those outcomes)
  - the effects the execution of the process has
  - composite processes (sequence, split, join, choice, ...)

# OWL-S: Process Model

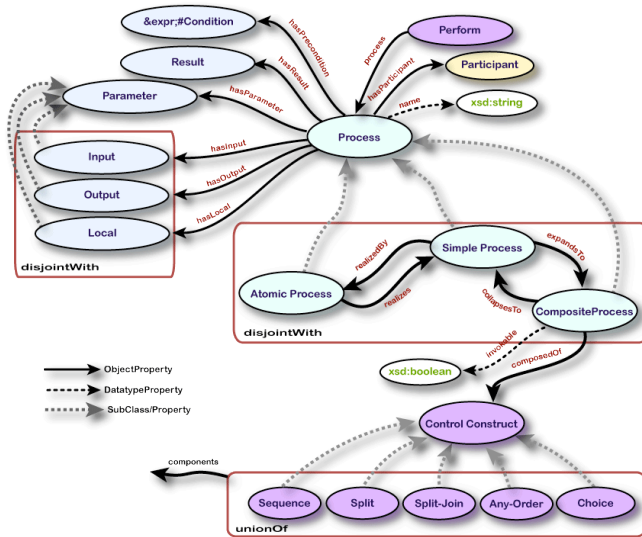


Figure: Extract of the Process ontology that is part of the OWL-S ontology.

# OWL-S: Process Model – composed processes

Example of a *Sequence* of processes where the second process (*Process2*) depends on data from the first process (*Process1*):

```
<process:Sequence rdf:ID="CP">
  <process:components rdf:parseType="Collection">
    <process:Perform rdf:ID="Step1">
      <process:process rdf:resource="#Process1"/>
      ...
    </process:Perform>
    <process:Perform rdf:ID="Step2">
      <process:process rdf:resource="#Process2"/>
      <process:hasDataFrom>
        <process:Binding>
          <process:theParam rdf:resource="#inputParameterOfProcess1"/>
          <process:valueSource>
            <process:ValueOf>
              <process:theParam rdf:resource="#outputOfProcess1"/>
              <process:fromProcess rdf:resource="#Step1"/>
            </process:ValueOf>
          </process:valueSource>
        </process:Binding>
      </process:hasDataFrom>
    </process:Perform>
  </process:components>
  ...
</process:Sequence>
```

- The grounding provides the required details about transport protocols (each instance of the class *Service* refer to a *ServiceGrounding* via the *supports* property).
- Main challenge tackled by the service grounding: *Invocation*
- Main function of the OWL-S grounding:  
Specify how the (abstract) inputs and outputs of an atomic process are to be realized concretely as messages, which carry those inputs and outputs in some specific transmittable format.
- issues described in a service grounding:
  - communication protocol
  - message formats
  - port numbers used in contacting the service
  - serialization: for each semantic type of input or output specified in the *ServiceModel*, the service grounding has to specify an unambiguous way of exchanging data elements of that type with the service (that is, the *serialization techniques* employed)

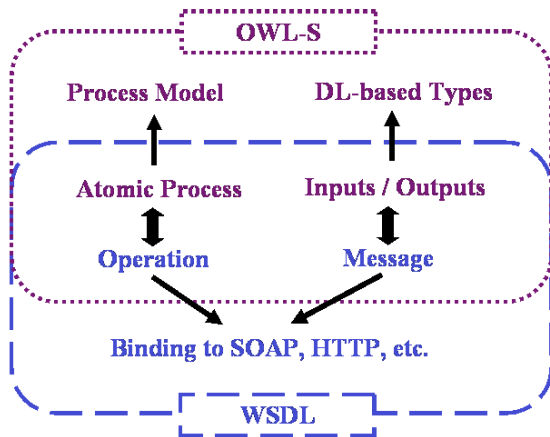


Figure: OWL-S/WSDL grounding: Mapping between OWL-S and WSDL.

## *Service Repository:*

- **Registry:** publishing and localization of Semantic Web services

## *Service Discovery and Selection:*

- **Web Service Discovery:** The given goal definition (possibly described by means of an OWL-S profile) is used to find Web service definitions in the repository which can fulfill that goal.
- **Service Selection:** Additional criteria like QoS and user preferences are used to rank the discovered services.

## *Execution:*

- **Process Mediator** (Orchestration / Choreography): resolves process heterogeneity between requester and provider.
- **Data Mediator:** resolves data heterogeneity between requester and provider.
- **Invoker:** mediates between service requester and provider for invoking a service.

## 1 Web Services

- Services on the Web
- Service Oriented Architectures
- Standards for Web Services
  - SOAP
  - WSDL
  - UDDI - Universal Description, Discovery and Integration
- Semantic Web Services
  - Why do we need Semantic Web Services?
  - Semantic Web Services – Challenges
  - OWL-S
  - Semantic Web Services Architecture

For more details on Web services and Service-oriented architectures:  
Daniel Lübke's lecture about "Entwicklung service-orientierter  
Architekturen und Anwendungen"

[http://www.se.uni-hannover.de/lehre/2008winter/ws2008\\_soa.php](http://www.se.uni-hannover.de/lehre/2008winter/ws2008_soa.php)