We introduce OWL-S. This talk is based on:

- The OWL Services Coalition. [OWL-S Technical Overview](#)

## 1 Introduction

- WSDL is a simple standard for describing webservices. It provides functionality similar to an API.
- If we hope to have agent compose just-in-time services from individual components, we will need more semantically-rich descriptions of services.
- That is, an agent needs to have some understanding of what `getStockQuote (string symbol)` does.
- OWL-Services is a set of ontologies, written in OWL, which can be used to describe (at a higher/more detailed semantic level) what a service does.
- Previously known as DAML-S.

## 2 Upper Ontology for Services

- Resources are available out in the net.
- The OWL-S ontology defines a Service as the central class for describing interfaces, part of the OWL-S [Service Ontology](#).
- *What does the service require and provide for the users?* This is given by the ServiceProfile. An agent uses it to determine whether the service meets it’s needs.
- *How is it used?* Given by the Service Grounding. It tells how to access the service.
3 Service Profiles

- The ServiceProfile ontology supports the use of three types of information.
  2. A specification of the functionalities that are provided by the service.
  3. Attributes which provide additional information and requirements (e.g., quality guarantees, expected response, geographic constraints, etc.)

- The functionalities are specified by declaring the IOPEs:
  - The Inputs the service expects.
  - The Output information returned.
  - The Preconditions that have to be satisfied in order to use the service.
  - The expected Effects from running the service.

3.1 ServiceProfile Ontology

3.2 Profile Description Attributes

  - **serviceName** is the name (ID).
  - **intendedPurpose** tells what constitutes successful accomplishment of service execution.
  - **textDescription** English description.
  - **providedBy** who provides it.
  - **requestedBy** who requests this service.

3.3 Functional Description Attributes

  - These attributes describe the interface.
  - **input** describes the input(s) the service can receive.
  - **output**
  - **precondition** describes what must be true in order to use the service.
  - **effect** what will happen when the service runs.
3.4 Functional Attributes

- A collection of other attributes that the service might have which do not deal with the process that the service implements.
- geographicRadius
- degreeOfQuality
- serviceParameter
- communicationThru
- serviceType
- serviceCategory
- qualityGuarantees
- qualityRating

4 Service Model

- Services are viewed as processes which are defined using a Process Ontology.
- A process can have any number of inputs.
- It can have any number of outputs.
- It has a parameter that specifies the participants in the process.
- It can have any number of preconditions that must hold for the process to be invoked.
- It can have any number of effects.
- Outputs and effects can have conditions associated with them.

4.1 Process Ontology

- An AtomicProcess is directly invocable, has not sub-processes, and executed in a single step.
• A **SimpleProcess** is not invocable (not associated with a grounding). Its executed in a single step. Used as an element of abstraction.

• A **CompositeProcess** is decomposable into other process using control constructs. It is composedOf a **ControlConstruct** which, in turn, has a **components** property that indicates the ordering and conditional execution of the sub-processes.

### 4.2 Control Constructs

• A **Sequence** is a list of **Processes** to be done in order.

• A **Split** contains a bag of process components to be executed concurrently.

• **Unordered** specifies a bag of process components that can be executed in any order.

• **Split+Join** consists of concurrent execution of process components with barrier synchronization.

• A **Choice** has further properties chosen and chooseFrom which let you create customized subsets.

• The **If-Then-Else** class has properties ifCondition, then, and else, which implement the statement.

• **Iterate** does just that until the whileCondition or untilCondition are met.

• **Repeat-Until** does a similar job.

### 4.3 Process Control Ontology

• It’s an ontology that represent methods for monitoring and controlling the progress of an executing process.

• It does not exist yet.

### 4.4 Time Ontology

• OWL-S also defines a simple **Time Ontology**.

• It has two main classes: **Instants** and **Intervals**.

• It has three properties that go from Interval to Instant:
  - start-of
  - end-of
  - inside

### 5 Resources

• There is also a **Resource Ontology**.

• Processes generally require(consume) resources.

• **Resources** have an **AllocationType** property which can be used to tell if the resource is consumable (e.g., time) or reusable (e.g., paint).
6 Congo Example

- This example is from the walkthrough.
- Congo is a website that sells books.
- Their services are LocateBook, PutInCart, SignIn, CreateAcct, CreateProfile, LoadProfile, SpecifyDeliveryDetails, FinalizeBuy.
- You can get the complete Congo example file set.

6.1 Describe the Program

- Congo offers the CongoBuy service which is composed of smaller programs.
- You should describe these programs first.
- These individual programs are defined as Process.

6.2 Process Input and Output

- The process ontology shows the various types of processes we can have.
- The LocateBook service is atomic, so we declare it as such:

```
<process:AtomicProcess rdf:ID="LocateBook">
  <process:hasInput>
    <process:Input rdf:ID="BookName">
      <process:parameterType rdf:resource="&xsd;#string"/>
    </process:Input>
  </process:hasInput>
  <process:hasOutput>
    <process:ConditionalOutput rdf:ID="LocateBookOutput">
      <process:coCondition rdf:resource="#InCatalogueBookInstance"/>
      <process:parameterType rdf:resource="LocatedBookOutput"/>
    </process:ConditionalOutput>
  </process:hasOutput>
</process:AtomicProcess>
```

- This also says that LocateBook takes as input a BookName, which is a string
- The output is conditional. If #InCatalogueBookInstance then return LocatedBookOutput.

6.3 Process Preconditions and Effects

- In order to tie a bunch of processes together (compose) we also need to know their preconditions for execution and any side-effects they might have.
- So, OWL-S also has precondition and effect (yes, like AI planner operators. 1970’s AI research might yet find an application :-)).
- ExpressCongoBuy service has two preconditions: you must have an account and credit:

```
<process:AtomicProcess rdf:ID="ExpressCongoBuy">
  <process:hasInput>
```

5
6.4 Composite Processes

- A CompositeProcess is composedOf a bunch of ControlConstructs which can be things like sequence, if-then-else, fork, while, etc.
- Build them in a top-down manner.
- FullCongoBuy has two steps: locating the book and then buying the book.
<!- All of the inputs and outputs of this composite process are derived from the corresponding inputs and outputs of its atomic processes and will normally be computed automatically by OWL-S tools. -->

<process:hasInput>
  <process:Input rdf:ID="FullCongoBuyBookName">
    <process:parameterType rdf:resource="&xsd;#string"/>
  </process:Input>
</process:hasInput>

<process:hasInput>
  <process:Input rdf:ID="FullCongoBuySignInInfo">
    <process:parameterType rdf:resource="#SignInData"/>
  </process:Input>
</process:hasInput>

<process:hasInput>
  <process:Input rdf:ID="FullCongoBuyCreateAcctInfo">
    <process:parameterType rdf:resource="#AcctInfo"/>
  </process:Input>
</process:hasInput>

<process:hasInput>
  <process:Input rdf:ID="FullCongoBuyCreditCardNumber">
    <process:parameterType rdf:resource="&xsd;#decimal"/>
  </process:Input>
</process:hasInput>

<process:hasInput>
  <process:Input rdf:ID="FullCongoBuyCreditCardType">
    <process:parameterType rdf:resource="#CreditCardType"/>
  </process:Input>
</process:hasInput>

<process:hasInput>
  <process:Input rdf:ID="FullCongoBuyCreditCardExpirationDate">
    <process:parameterType rdf:resource="&xsd;#string"/>
  </process:Input>
</process:hasInput>

<process:hasInput>
  <process:Input rdf:ID="FullCongoBuyCreditCardDeliveryAddress">
    <process:parameterType rdf:resource="&xsd;#string"/>
  </process:Input>
</process:hasInput>

<process:hasInput>
  <process:Input rdf:ID="FullCongoBuyPackagingSelection">
    <process:parameterType rdf:resource="&xsd;#string"/>
  </process:Input>
</process:hasInput>

<process:hasInput>
  <process:Input rdf:ID="FullCongoBuyDeliveryTypeSelection">
    <process:parameterType rdf:resource="&xsd;#string"/>
  </process:Input>
</process:hasInput>
7 Conclusion

- OWL-S is more complex than WSDL.
- OWL-S gives a lot more details about how a process is composed of other process, what sequence they must execute, etc.
- It’s processes are akin to AI-planning operators.
- Just-in-time service composition will be much more likely if services are described using OWL-S. Unfortunately that will require extra effort on the programmer’s part (WSDL can be generated automatically).
- OWL-S is sits righ between web-services (RPCs over HTTP) and the Semantic Web vision.

Notes

1. http://www.daml.org/services/owl-s/1.0/owl-s.html
3. http://www.daml.org/services/owl-s/1.0/Service.owl
4. http://www.daml.org/services/owl-s/1.0/Profile.owl
5. http://www.daml.org/services/owl-s/1.0/Process.owl
7. http://www.daml.org/services/owl-s/1.0/Resource.owl
8. http://www.daml.org/services/owl-s/1.0/examples.html

This talk is available at http://jmvidal.cse.sc.edu/talks/owls

Copyright © 2004 Jose M Vidal. All rights reserved.