AOT Lab

OWL-S

Semantic Markup for Web Services

See: *http://www.w3.org/Submission/2004/SUBM-OWL-S-20041122/*

AOT Lab

What is OWL-S?

- OWL-based Web service ontology
  - Supplies a core set of markup language constructs for describing Web services in unambiguous, computer-interpretable form
    - Describe Web services capabilities
    - Describe Web services Process Model
    - Map Web services Process Model to WSDL for Web service invocation
  - OWL-S allows services to interact on the Semantic Web
    - Description of capabilities allows capability-based discovery of WS
    - Process Model allows construction of plans that compose the activities of different WS
    - Mapping to WSDL allows automatic invocation of WS
  - OWL-S objective
    - OWL-S does not aim to replace the Web services standards rather it attempts to provide a semantic layer
      - OWL-S relies on WSDL for Web service invocation
      - OWL-S expands UDDI for Web service discovery

AOT Lab

Motivation

- Tasks OWL-S is expected to enable:
  - ***Automatic Web service discovery***
    - Automated location of WSs that provide a particular service and adhere to requested constraints
  - ***Automatic Web service invocation***
    - Automated execution of an identified WS by a computer program or agent
  - ***Automatic Web service composition and interoperation***
    - Automatic selection, composition and interoperation of WSs to perform some task (e.g. arrangement for a conference)
  - ***Automatic Web service execution monitoring***
    - Individual services and composition services generally require some time to execute completely
    - It is useful to know the state of execution of services

AOT Lab

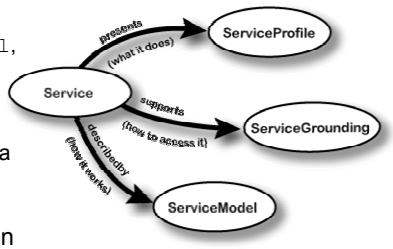
Upper Ontology for Services

- Three essential type of knowledge about a service:
  - ***What does the service provide for prospective clients?***
    - The answer to this question is given in the "profile" which is used to advertise the service.
    - To capture this perspective, each instance of the class `Service` presents a **`ServiceProfile`**.
  - ***How is it used?***
    - The answer to this question is given in the "process model"
    - This perspective is captured by the `ServiceModel` class. Instances of the class `Service` use the property describedBy to refer to the service's **`ServiceModel`**.
  - ***How does one interact with it?***
    - The answer to this question is given in the "grounding"
    - A grounding provides the needed details about transport protocols. Instances of the class `Service` have a supports property referring to a **`ServiceGrounding`**.

Source: http://www.w3.org/Submission/2004/SUBM-OWL-S-20041122/

Upper Ontology for Services

- The class `Service` provides an organizational point of reference for a declared Web service
  - One instance of `Service` will exist for each distinct published service.
  - The properties *presents*, *describedBy*, and *supports* are properties of `Service`.
  - The classes `ServiceProfile`, `ServiceModel`, and `ServiceGrounding` are the respective ranges of those properties.
  - Each instance of `Service` will *present* a `ServiceProfile` description, be *describedBy* a `ServiceModel` description, and *support* a `ServiceGrounding` description.
- The `ServiceProfile` provides the information needed to automatically discover a service, while the `ServiceModel` and `ServiceGrounding`, taken together, provide enough information to make use of a service, once found



Source: http://www.w3.org/Submission/2004/SUBM-OWL-S-20041122/

Upper Ontology for Services

- The upper ontology for services specifies only <u>two cardinality constraints</u>:
  - <u>A service can be described by at most one service model</u>
  - <u>A grounding must be associated with exactly one service</u>.
- It deliberately does not specify any minimum cardinality for the properties *presents* or *describedBy*
  - In principle, a service needs all three properties to be fully characterized; in some situations a partial characterization could be useful.
- Nor does it specify any maximum cardinality for *presents* or *supports*
  - It can be useful for some services to offer multiple profiles and/or multiple groundings.

Service Profile

- The class `ServiceProfile` provides a superclass of every type of high-level description of a service
- OWL-S provides one possible representation of a service profile through the class `Profile`, describing a service as a function of three basic types of information:
  - What organization provides the service
    - Contact information that refers to the entity that provides the service
  - What function the service computes
    - Specified in terms of:
      - Inputs required by the service and outputs generated
      - Preconditions required by the service and expected effects that result from the execution of the service
  - A host of features that specify characteristics of the service
    - The category of a given service
    - The quality rating of the service (some services may be very good, reliable, and quick to respond)
    - An unbounded list of service parameters that can contain any type of information

Service Profile and Registry

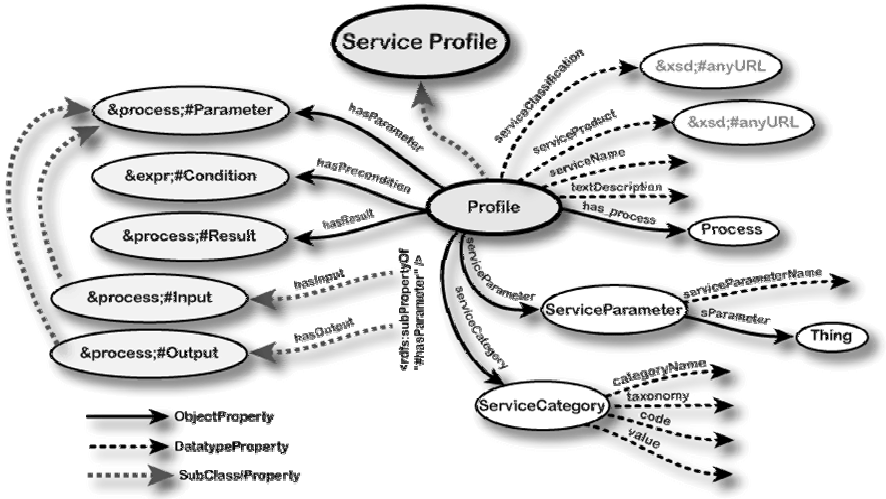
- The aim of the service profile is to provide a concise description to a registry
- The types of registry may vary widely
  - 28 different types have been identified
  - By using a declarative representation of Web services, the service profile is not committed to any form of registry
    - It can be used in all of them.
  - The service profile can also be used to represents needs of services
    - In a reverse registry that records needs and queries on offers.

Service Profile vs. Service Model

- The Profile and the Process Model play different roles during the transaction between Web services
  - **But …**

    they are two different representations of the same service and the input, output, precondition, and effects (IOPEs) of one are reflected in the IOPEs of the other
- OWL-S does not dictate any constraint between Profiles and Process Models
  - … the two descriptions may be inconsistent without affecting the validity of the OWL expression

Profile Properties



- Selected class and properties of the `Profile`

Source: http://www.w3.org/Submission/2004/SUBM-OWL-S-20041122/

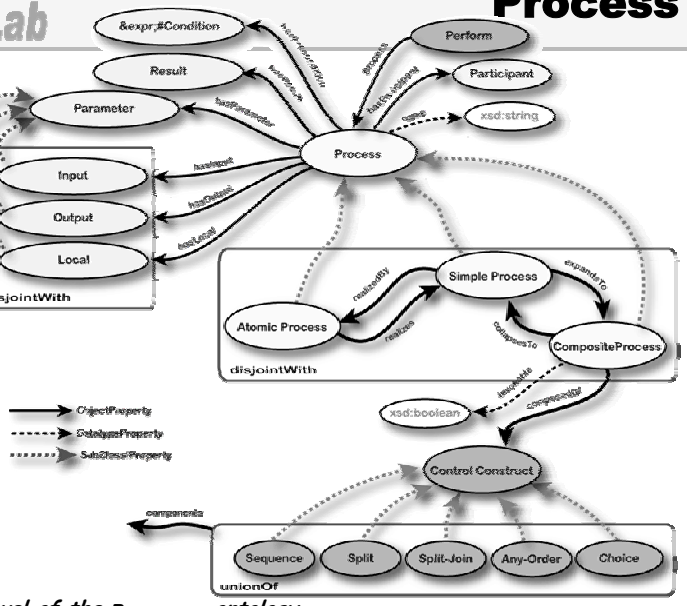
AOT Lab Profile - Functionality Description

- Generally the IOPE's published by the `Profile` are a subset of those published by the `Process`.
  - The `Process` part of a description will create all the IOPE instances and the `Profile` instance can simply point to these instances.
  - The `Profile` can create its own IOPE instances according to the schema in the Process ontology
- The Profile ontology defines the following properties of the Profile class:
  - ***hasParameter***
    - ***hasInput***
    - ***hasOutput***
  - ***hasPrecondition***
  - ***hasResult***
    - Specifies under what conditions the outputs are generated and what domain changes are produced during the execution of the service

AOT Lab Profile - Additional Properties

- **serviceParameter**
  - An expandable list of properties that may accompany a profile description.
  - The value of the property is an instance of the class `ServiceParameter`
    - **serviceParameterName**
      - Name of the actual parameter (e.g. the URI)
    - **sParameter**
      - Points to the value of the parameter within some OWL ontology.
- **serviceCategory**
  - Describes categories of services on the bases of some classification.
  - The value of the property is an instance of the class `ServiceCategory`
    - **categoryName**
    - **taxonomy**
      - A reference to the taxonomy scheme (not necessarily an URL).
    - **value**
      - Points to the value in a specific taxonomy
    - **code**
      - Code associated to a taxonomy.

Process Model



- Top level of the `Process` ontology

Source: http://www.w3.org/Submission/2004/SUBM-OWL-S-20041122/

Modelling Services as Processes

- A process is intended as a specification of the ways a client may interact with a service
  - An *atomic* process is a description of a service that expects one (possibly complex) message and returns one (possibly complex) message in response.
  - A *composite* process is one that maintains some state; each message the client sends advances it through the process.
- A process can have two sorts of purpose:
  1. It can generate and return some new information. **Information** production is described by the inputs and outputs of the process.
  2. It can produce a change in the **world**. This transition is described by the preconditions (which must all hold in order for the process to be successfully invoked) and effects of the process
     - Preconditions and effects are represented as logical formulas (using languages more expressive than OWL: RuleML or OWL Rules Language)

Simple Process

- ***Simple*** processes are not invocable and are not associated with a grounding
  - They are conceived of as having single-step executions
- Simple processes are used as elements of abstraction. A simple process may be used to provide:
  - A view of (a specialized way of using) some atomic process
    - The simple process is ***realizedBy*** the atomic process
  - A simplified representation of some composite process (for purposes of planning and reasoning).
    - The simple process ***expandsTo*** the composite process

Composite Process

- ***Composite*** processes are decomposable into other (non-composite or composite) processes
  - Their decomposition can be specified by using control constructs
  - A process can often be viewed at different levels of granularity, either as a primitive, undecomposable process or as a composite process.
    - These are sometimes referred to as ``black box'' and ``glass box'' views, respectively.
    - When a composite process is viewed as a black box, a simple process can be used to represent it.
- A composite process is not a behavior a service *will* do, but a behavior (or set of behaviors) the client *can* perform by sending and receiving a series of messages.
  - If the composite process has an overall effect, then the client must perform the entire process in order to achieve that effect.

Process - Control Constructs

- **`Sequence`**
  - A list of control constructs to be done in order.
- **`Split`**
  - A bag of process components to be executed concurrently. Split completes as soon as all of its component processes have been scheduled for execution.
- **`Split-Join`**
  - The process consists of concurrent execution of a bunch of process components with barrier synchronization. Split+Join completes when all of its components processes have completed.
- **`Any-Order`**
  - Allows the process components (specified as a bag) to be executed in some unspecified order but not concurrently. *Execution and completion of all components is required*.
- **`Choice`**
  - Execution of a single control construct from a given bag of control constructs. Any of the given control constructs may be chosen for execution
- **`If-Then-Else`**
- **`Iterate`**
  - Is an "abstract" class, serves as the common superclass of Repeat-While, Repeat-Until, and potentially other specific iteration constructs.
  - **`Repeat-While`** and **`Repeat-Until`**
    - Iterate until a condition becomes false or true, following the familiar programming language conventions.

Data Flow and Parameter Bindings

- In composite processes we can have different type/pattern of ***data flow*** specifications
  - The input to one process component can be obtained as one of the outputs of a preceding step.
  - The outputs of a composite process may be derived from outputs of some of its components
  - …
- The convention adopted is that the source of a datum is identified when the user of the datum is declared (***consumer-pull*** convention)
  - If step 1 feeds step 3, this fact is specified in the description of step 3 rather than in the description of step 1 (the opposite is called ***producer-push*** convention).

Data Flow and Parameter Bindings

- Ex.

I1 input of: { Composite Process CP }: with output O1 composed of
Step 1: Perform S1 ⇒ Step 2: Perform S2

where: S1 has inputs I11 and I12, and output O11
S2 has input I21 and output O21

  - Suppose that:
    - Input I1 of the overall process CP is used as input I11 of S1, after adding 1.
    - Input I12 of S1 is a constant, the string "Academic".
    - Output O11 of S1 is used as input I21 of S2.
    - The maximum of 0 and output O21 of S2, times π, is used as output O1 of CP.
  - Using a consumer-pull convention, the parameters I1, O11, and O21 are simply declared, but for parameters I11, I21, and O1 ***bindings*** are provided:

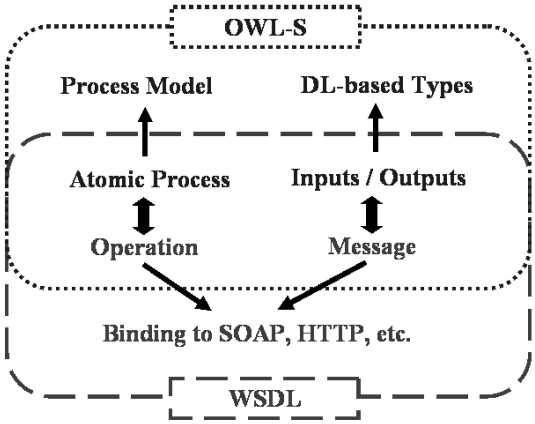
I11(Step1) comes from incr(I1(CP))
I12(Step1) = "Academic"
I21(Step2) comes from O11(Step1)
O1(CP) comes from π × max 0, O21(Step2))

  - Each equalities is represented in OWL-S as a ***Binding***, an abstract object with two properties: `toParam`, the name of the parameter (e.g., I21(S2)), and `valueSpecifier`, a description of its value.

Grounding a Service

- Providing details on how to interoperate/access the service
  - Protocol, message formats, serialization, …
  - A mapping from an abstract specification to a concrete realization
    - How the abstract inputs and outputs of an atomic process are to be realized concretely as messages (which carry these inputs and outputs)
- WSDL as a possible grounding approach
  - Exploiting the extensibility elements of WSDL

OWL-S/WSDL Grounding



- To construct an OWL-S/WSDL grounding one must first identify, in WSDL, the messages and operations by which an atomic process may be accessed, and then specify correspondences

OWL-S and UDDI

- The main problem with UDDI is that it does not provide a **capability representation language** such as the OWL-S Service Profile.
  - ➔ UDDI supports the location of information about a Web services, once it is known which Web service to use
  - ➔ **UDDI does not provide capability based search** (impossible to locate a Web service on the basis of what problems it solves)
- **But** …

  OWL-S and UDDI complement each other

  - ➔ Integrate OWL-S capability matching in the UDDI registry.
    - Mapping of OWL-S Service Profiles into UDDI Web service representations.
    - A set of specialized UDDI TModels to store OWL-S information that cannot be represented in the standard UDDI
  - OWL-S/UDDI provides all the functionalities provided by UDDI using exactly the same API; any UDDI can interact with it to retrieve information about available Web services.
  - OWL-S/UDDI supports capability matching by taking advantage of OWL-S capability representation.

OWL-S to UDDI Mapping