Semantic Web Services and the OWL-S framework

Technical Report in fulfillment of the thesis requirement for the degree of PhD “Ontological Reasoning for Learning Services in the Semantic Web”

TR: MRXP2

Maria Tsiakmaki

Committee: Omiros Ragos* Chrisafis Hartonas† Panagiotis Pintelas†

University of Patras
Department of Mathematics
GR-265 04 Patras, Greece

2011

*Department of Mathematics, University of Patras, Greece
†Department of Computer Science and Telecommunications, TEI of Larissa, Greece
Abstract

The proposed research focuses on the context of learning in Semantic Web. In such an environment basic entities exist, interact and are changing: Learning Objects, the containers of educational materials, Learners, users whose goal is to enhance their learning portfolio, and Learning Services, the processes that deliver Learning Objects to Learners and manage the interaction between Learning Objects and Learners.

Learning Objects may be atomic (unstructured, free-standing) or compound (recursively containing other Learning Objects), static (structured at design time) or dynamic (structured at run-time), they should be reusable, tailored to Learner needs, affordable and discoverable.

Learners come with a knowledge level, preferences and a learning style, while their goal is to learn.

Learners interact with Learning Objects in a physical/ technical level (e.g. navigate), in a linguistic level (e.g. Learner–service dialog) and in a modeling level (e.g. activities related to modeling the knowledge, comprehension of the Learner, or modeling the flexibility and the adaptable features of the service).

In a network where reasoning can be implemented by computers, that is the Semantic Network, technologies and methodologies such as Ontological Engineering, Logic and Inference Rules, Description Logic and Intelligent Agents, can be profitably used in addressing and resolving a number of issues related to the context of this research. We will concentrate on (a) the ontologically–driven Learner and Service modeling, and (b) the ontologically–driven management of the Learner–Service dialog. Related issues are also (a) the reference architecture for Learning Objects and Services, (b) Learner/ Service Modeling Services and Dialog Management Processes, and (b) standardization of a semantic annotation (metadata) to describe a Learner model and a Service model (e.g. the IEEE LOM standard).
Contents

1 Preface 1

2 Semantic Web 1
  2.1 The Semantic Web Layer Cake 1

3 Web Services 3
  3.1 SOAP 3
  3.2 WSDL 4
  3.3 UDDI 5

4 Semantic Web Services 5
  4.1 Semantic Web Service Ontologies 6
  4.1.1 OWL-S 6
  4.1.2 The Upper Ontology for Services 6
  4.1.3 The Upper Ontology for Find Cheaper Book Service 7
  4.1.4 Service Profile 7
  4.1.5 Find Cheaper Book Profile 9
  4.1.6 Process Model 9
  4.1.7 Find Cheaper Book Process Model 17
  4.1.8 Grounding 17
  4.1.9 Some criticisms related to Learning Services 18
  4.2 WSMO 20

5 Tools 20
  5.1 Protege 20
  5.2 Some Conclutions and future work 21

Bibliography 22

Appendix 22

A Acronyms and Abbreviations 23

B WSDL document Example 23

C OWL-S Expression Examples 24

D OWL-S Process Example 26
## List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Semantic Web Layers</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>Web Services Architecture Stack</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>UDDI core data</td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td>OWL-S Upper Ontology</td>
<td>7</td>
</tr>
<tr>
<td>5</td>
<td>OWL-S Upper Ontology Find Cheaper Book</td>
<td>7</td>
</tr>
<tr>
<td>6</td>
<td>OWL-S Service Profile</td>
<td>8</td>
</tr>
<tr>
<td>7</td>
<td>OWL-S Expression</td>
<td>10</td>
</tr>
<tr>
<td>8</td>
<td>OWL-S Process Properties</td>
<td>11</td>
</tr>
<tr>
<td>9</td>
<td>OWL-S Result</td>
<td>11</td>
</tr>
<tr>
<td>10</td>
<td>OWL-S Process and sub-categories</td>
<td>12</td>
</tr>
<tr>
<td>11</td>
<td>Find Cheaper Book – Composite Process</td>
<td>17</td>
</tr>
<tr>
<td>12</td>
<td>Find Cheaper Book – Compare Prices</td>
<td>18</td>
</tr>
<tr>
<td>13</td>
<td>Mapping between OWL-S and WSDL</td>
<td>19</td>
</tr>
<tr>
<td>14</td>
<td>OWL-S Grounding</td>
<td>19</td>
</tr>
<tr>
<td>15</td>
<td>The main elements of WSMO</td>
<td>20</td>
</tr>
</tbody>
</table>
1 Preface

The domain of our discourse is Learning Services in the Semantic Web. For this, in this technical report we will introduce the environment in which these services are expected to live in, i.e. the Semantic Web, and the technologies/standards that are being implemented in such environments, i.e. RDF and OWL.

Learning Services are Semantic Web Services. Thus we secondly discuss about Web Services and their standards (UDDI, WSDL, SOAP).

Finally, we detail two basic proposals that use ontologically driven representations for Web Services: OWL-S and WSMO.

2 Semantic Web

Since the beginning of the Internet we can roughly discriminate three different eras: Web 1.0 (1990-2000) and Web 2.0 (2000-now), while we are visioning Web 3.0 or else the Semantic Web. During Web 1.0 there was tendency of post on-line every available information. Books, news, personal web pages, encyclopedias, images, music became in digital format, creating a majority of web pages. Web 1.0 gave birth to the “dotcom” companies and technologies, such as HTML and hypertext.

While this trend has not stopped (and probably will never stop as new data become available) more needs arose, like the effective management, organizing and integration of content, personalization and customization of content. During Web 2.0, content is more dynamic and end-users are powered to customize, generate and modify content for themselves (e.g. social networks, blogs, video broadcasting, wikis, podcasts, review services and products, social networks, RSS,...). For this, more advanced information management technologies, computational linguistics, statistical learning theories are been developed and used.

But since now, the structure of the web content is only suitable for human consumption. The basic idea of the Semantic Web is to make content machine readable, so as machines can reason on it. Although current technologies provide an advanced level of services, they still seem limited. For instance, even though search engines are very effective in retrieving text, videos, images, a lot of work remains to be done for answering logical questions, e.g. it is very difficult to find an article written by George W. Bush, than articles written about George W. Bush. Semantic Web promises to “unleash a revolution of new possibilities” [2].

2.1 The Semantic Web Layer Cake

For a new approach to be utilized and established, one of the first things is to decide on its architecture. The same stands for Semantic Web: some components need to be clarified so as to be further investigated. The next thing is to specify some standards, i.e. technologies to materialize the vision. In spring 2006, W3C concluded on the acceptance of three standards: RDF, RDFS and OWL.

The above diagram (Figure 1) illustrates a layered model with the key technologies that builds Semantic Web from the syntactic, through the semantic and logic, to higher level concepts such as proof and trust [1].
URI
URI stands for Uniform Resource Identifier, a compact, unique characters string used to identify or name a resource, e.g. a website’s URL.

Unicode
Unicode is the universal standard encoding system and provides a unified system for representing any character in any language.

XML
XML stands for Extensible Markup Language. It’s a language similar to HTML that enables users to define and use their own elements (tags). Each element may have attributes, or contain other elements. With XML we can structure our data, and share them across the web. The functional meaning of the elements depends on the nature of the application.

XML Namespaces
XML Namespaces are like a vocabulary of named elements and attributes that can be used in an XML document. It is mechanism for grouping elements in order to resolve ambiguities between identically named elements or attributes across XML documents.

XML Schema
XML Schema describes the structure of XML documents. It is a set of rules that someone needs to abide by when writing an XML document.

RDF
RDF stands for Resource Description Framework. It is a framework for describing the relationships among resources (URIs) in terms of named properties and values. It has an XML-based syntax that enables someone to write simple statements. A statement in RDF is called a triple (Subject-Predicate-Object → Resource-Property-Value).

RDF Schema
RDF Schema describes the structure of RDF documents. It describes the way we have to define resources, and properties in an RDF document.

Ontology
Ontologies extend RDF and RDFS in order to describe more complex relations among resources of a domain. It contains a taxonomy of concepts and a set of inference rules. A taxonomy classifies concepts hierarchically (classes and subclasses) and defines relationships and constraints among them (properties). Inheritance is enabled. Inference rules restrict further the described domain,
increase its validity and consistency and enable reasoning activities.

The Web Ontology Language (OWL) is used in this layer. There are three increasingly-expressive versions of OWL: OWL Lite, OWL DL, and OWL Full. With SWRL and F-Logic we can express rules.

**Logic**
In the logic layer, rules are used to make inferences.

**Proof**
The proof layer involves the actual deductive process as well as the representation and validation of proofs in Web.

**Digital Signatures**
The Digital Signatures can be used in Semantic Web to authenticate the origin of and ontology and of a deduction.

**Trust**
The trust layer will label recourses and deductions as trusted bases on recommendations by trusted agents or on rating and certification agencies.

## 3 Web Services

A Web Service is defined as “a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-process able format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards.” [3]

Web Services support Web standards for creating, describing, advertising and locating other Web Services. Below (Figure 2) lies a short description of SOAP, WSDL and UDDI frameworks.

![Web Services Architecture Stack](image)

**Figure 2: Web Services Architecture Stack**

### 3.1 SOAP

Simple Object Access Protocol (SOAP) is “a lightweight protocol intended for exchanging structured information in a decentralized, distributed environment.
It uses XML technologies to define an extensible messaging framework providing a message construct that can be exchanged over a variety of underlying protocols." [8] [9]

The specification of this protocol describes the

1. SOAP messaging framework
   - the rules for processing a SOAP message
   - the concepts of SOAP features and SOAP modules
   - the binding framework, i.e. the rules for defining a binding to an underlying protocol that can be used for exchanging SOAP messages between SOAP nodes
   - the structure of a SOAP message

2. the features of the SOAP, and

3. adjuncts that can be used in connection with the SOAP messaging framework

3.2 WSDL

Web Service Description Language (WSDL) is an “XML format for describing network services as a set of endpoints operating on messages containing either document-oriented or procedure-oriented information. The operations and messages are described abstractly, and then bound to a concrete network protocol and message format to define an endpoint. Related concrete endpoints are combined into abstract endpoints (services).” [4]

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. The concrete protocol and data format specification 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. 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).
- Message – an abstract, typed definition of the data being communicated.
- Operation – an abstract description of an action supported by the service.
- Port Type – an abstract set of operations supported by one or more endpoints.
- Binding – a concrete protocol and data format specification for a particular port type.
- Port – a single endpoint defined as a combination of a binding and a network address.
- Service – a collection of related endpoints.

In Appendix B there is an example WSDL document
3.3 UDDI

Universal Description, Discovery and Integration (UDDI) [15] is a protocol that defines a standard method for describing, publishing and discovering Web Services. Service providers describe in a XML file information that users and applications should need in order to use the particular service. Afterwards they publish this file in an UDDI registry.

The core information contained is such a file is (Figure 3):

- information about the organization/business that offers the service (businessentity)
- description of the service’s business function (businessservice)
- technical details about the services (bindingTemplate)
- various attributes or metadata about the service, such as taxonomy, transports, digital signatures (tModels)
- relations among entities (publisherAssertions)
- standing requests to track changes to a list of entities (subscription)

Figure 3: UDDI core data

4 Semantic Web Services

Semantic Services are a component of the Semantic Web and they combine Web Services and Semantic Web technologies:

- Semantic Web adds machine readable semantics to data, so that machines can “understand” the information and reason on it, and
- Web Services employ the Web as a global infrastructure for distributed computation, for integrating various applications and for the automation of business processes.

The XML standards for the interoperability of Web Services (UDDI, WSDL, SOAP) specify only syntactic interoperability (a pre-defined syntax), not the semantic meaning of messages. For example, Web Services Description Language (WSDL) can specify the operations available through a Web Service and
the structure of data sent and received but cannot specify semantic meaning of the data or semantic constraints on the data, interpret and reason on them. This requires programmers to make specific agreements on the interaction of web services and makes the automatic Web Service composition difficult.

Semantic Web is built upon universal standards (RDF, RDFS, OWL) for the interchange of semantic data, which makes it easy for programmers to combine data from different sources without losing meaning.

Semantic Web Services will enable the automation of services usage tasks such as:

- automatic discovery and selection of appropriate Web Services
- automatic invocation (execution) of a Web Service
- automatic composition and inter-operation of Web Services
- automatic monitoring the execution of a Web Service

4.1 Semantic Web Service Ontologies

Semantic Web Service Ontologies aim to describe Web Services in the Semantic Web. Research to these fields has lead to the development of various frameworks, such as OWL Service Ontology (OWL-S) [13], Web Service Modeling Ontology (WSMO) [14], First Order Logic for SWS (FLOWS), METEOR-S, IRS-III... Next, we will discuss OWL-S and WSMO.

4.1.1 OWL-S

OWL-S (formerly DAML-S) [13] is an ontology for describing the properties and capabilities of a Web Service in unambiguous, computer-interpretable form. According to the Semantic Web layered approach (Figure 1), OWL-S is being built on the Ontology layer.

The overall structure of the OWL-S ontology contains three main parts: the Service Profile for advertising and discovering services; the Process Model, which gives a detailed description of a service’s operation; and the Grounding, which provides details on how to interoperate with a service, via messages.

While describing the parts of the ontology, some code parts of an OWL-S Service example will be given, hopefully, for a better understanding. Find Cheaper Book service is a price comparison service across two bookstores: for a given book title it searches its price in Amazon and Barnes and Nobles, and returns the cheapest result. Here we will display parts of the Find Cheaper Book Service. “This work was conducted using the Protg resource, which is supported by grant LM007885 from the United States National Library of Medicine” [7].

Illustrations are exported with the use of Protg [7] plugins OntoViz, OWLviz and OWL-S Editor [5].

4.1.2 The Upper Ontology for Services

The Upper ontology of Services includes three basic concepts (Figure 4):

ServiceProfile what does the Service require and provide, i.e. the capabilities of the service.
**ServiceModel** how the Service works, i.e. the description of what happens when the service is carried out.

**ServiceGrounding** how the Service is used, i.e. communication protocol, message formats, service’s port number....

![Diagram of OWL-S Upper Ontology](image)

**Figure 4: OWL-S Upper Ontology**

The Upper Ontology suggests two cardinality restrictions:

1. A Service is describedBy at least one ServiceModel
2. A ServiceGrounding supports exactly one Service

### 4.1.3 The Upper Ontology for Find Cheaper Book Service

Find Cheaper Book OWL-S Service looks like this (Figure 5):

```xml
<!-- Service description -->
<service:Service rdf:ID="FindCheaperBookService">
  <service:presents rdf:resource="#FindCheaperBookProfile"/>
  <service:describedBy rdf:resource="#FindCheaperBookProcess"/>
  <service:supports rdf:resource="#FindCheaperBookGrounding"/>
</service:Service>
```

![Diagram of OWL-S upper ontology for Find Cheaper Book Service](image)

**Figure 5: OWL-S Upper Ontology Find Cheaper Book**

### 4.1.4 Service Profile

The Service Profile tells “what the service does”, i.e. it gives the type of information needed by a service-seeking agent to determine whether the service meets its needs.

The Service Profile describes three basic types of information:
contact information  human readable information about the organization that provides the Service

functionality description  the Inputs and Preconditions required by the Service and the Outputs and Effects produced by the execution of the Service (IOPE)

further features, attributes of the Service  e.g. the category of the Service given a classification system, the Rating of the Service, the maximum response time, the geographic availability of the Service (deprecated in the 1.2 release).

There are no constraints related with the IOPEs of a Profile and Model. But if they are inconsistent, the interaction will break at some point.

The Profile’s IOPEs should be carefully selected from Model’s IOPEs: it may hide functionalities described in Model so as to be more general and may specify only the intended purposes of the Service.

Once a Service is selected the Profile becomes useless.

![Figure 6: OWL-S Service Profile](image)

Service Profile Properties  The above three sections classify the Service Profile properties (Figure 6):

presentedBy  the Service that presents

serviceName, textDescription, contactInformation  (e.g. an Actor)

hasParameter, hasInput, hasOutput, hasPrecondition, hasResult  functionalities that shows the information transformation (inputs and outputs), and the state change (preconditions, effects)
4.1.5 Find Cheaper Book Profile

Find Cheaper Book OWL-S Profile looks like this:

```xml
<profile:Profile rdf:ID="FindCheaperBookProfile">
  <service:isPresentedBy
    rdf:resource="#FindCheaperBookService/>
  <profile:serviceName xml:lang="en">
    Cheaper Book Finder
  </profile:serviceName>
  <profile:hasInput rdf:resource="#BookName"/>
  <profile:hasOutput rdf:resource="#BookInfo"/>
</profile:Profile>
```

4.1.6 Process Model

OWL-S describes the ways a Client may interact with the Service using Processes. The purpose of a Process is

1. to create new information (the Outputs), based on given information (the Inputs), and
2. to produce a change in the world (the Effects), depending on conditions that hold true while the Process is performed (the Preconditions).

A Process can be

atomic a single interaction, or
composite a multi-step action, or
simple an abstract atomic Process, used to provide multiple views of the same process

Some supporting Concepts In order to define Process, some supporting concepts are used.

ProcessVar ProcessVar is a subclass of SWRL Variable. It is a simple way to represent variables in OWL-S.

ProcessVar has one type and an optional value expression. It is the disjoint union of

Local a distinct type of variable used for intermediate results in a Composite Process.

Existential a variable that can occur in a precondition and a Result.

ResultVar

Participant the agents involved in the service, i.e. the Client and the Server.

Parameter Parameter is the disjoint union of Input and Output

1SWRL is the language for expressing rules in OWL
2OWL-S treats participants as Variables cause – as it is stated in the current release 1.2 - different values (participants) occur on different process calls
**Expression** A Process will not be executed properly unless its Preconditions are true. If and when it does execute, it has various Effects. Preconditions and Effects are represented as logical formulas. To represent formulas in RDF/OWL, OWL-S uses string literals (QuotedExpression e.g. in KIF, PDDL) or XML literals (UnquotedExpression e.g. SWRL, RDF).

As Conditions and Effects are both literal expressions, there is no way to discriminate them. Thus, OWL-S defines two separate sub-classes of Expression (Figure 7).

Figure 7: OWL-S Expression

Expression Examples are displayed in Appendix C

**OWL-S and scope of Variables** In RDF/OWL there is no notion of the scope of Variables. They are all treated as global. OWL-S wraps variable references inside literals to allow us to sneak in and impose our own scoping rules. Thus, the properties from a Process to its Parameters specify its scope: Participant, Input, Output, and Local parameters have as scope the entire Process they occur in. ResultVars are scoped to a particular Result, bound in the result’s condition, and used to describe the outputs and effects associated with that condition.

**OWL-S and Expressions - The XML Literal trick** RDF parser ignores literals. So, the put OWL-S parser to extract the ignored stuff and interpret it appropriately for its context, treating it as ordinary RDF after transformations such as replacing occurrences of variables with their values.

**Process Properties in detail** OWL-S connects Processes to their IOPEs using the following properties (Figure 8):

- hasParticipants: the Process involves at least two Participants (Process-Var), i.e. theClient, and theServer.
- hasInput: the input information that the process requires for its execution (Parameter)
- hasOutput: outputs that can be expected for this process (Parameter)
- hasPreconditions: the conditions that must be true in order the Process to be performed successfully (Condition)
Figure 8: OWL-S Process Properties

- hasResult: refers to a coupled Output and Effect (Result), binded with the Output of the Process (Result, see Figure 9).
  - inCondition: the Condition under which this Result (and not another) occurs
  - withOutput: the OutputBinding if the Condition is true
  - hasEffect: the changes of the state of the world are expressed as Effects
  - hasResultVar: variables bound to the inCondition, and scoped to the particular Result, and are used to describe the outputs and effects associated with that condition.

- hasExistential: variables bound to in Preconditions

Figure 9: OWL-S Result

In most cases the output and effect of a process varies according to Input and Preconditions. For example, if a process contains a step to buy an item,
there are two possible outcomes: either the purchase succeeds or it fails. In the former case, the effect is that ownership is transferred and the output is, say, a confirmation number. In the latter case, there is no effect, and the output is a failure message.

**Atomic and Simple Process** Atomic processes correspond to the actions a service can perform by engaging it in a single interaction. They take an input message, do something, and then return their output message. For each atomic process, there must be provided a grounding that enables a service requester to construct messages to the process from its inputs and deconstruct replies.

Simple processes are not invocable and are not associated with Grounding, but, like atomic processes, they are conceived of as having single-step executions. Simple processes are used as elements of abstraction; a simple process may be used either to provide a view of (a specialized way of using) some atomic process, or a simplified representation of some composite process (for purposes of planning and reasoning). In the former case, the simple process is realizedBy the atomic process; in the latter case, the simple process expandsTo the composite process (Figure 10).

**Composite Process** Composite processes are decomposable into other (non-composite or composite) processes. Their decomposition can be specified by using the control constructs.

The basic idea is that each Composite Process is composedOf a Control-Construct, which may be a Sequence, Split, Split + Join, Choice, Any-Order, Condition, If-Then-Else, Iterate, Repeat-While, and Repeat-Until.

Each such ControlConstruct, in turn, has a “components” property (a list or bag), which specifies the subcomponents of the ControlConstruct. These
classes are themselves control constructs.

Finally, when the components of a composite process are invocations ("performs") of atomic or composite processes, the Perform ControlConstruct is used.

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.

Computed Input, Output, effect and precondition  A computed input is a single expression that characterizes the inputs required by a composite process, and the conditions under which they are required. This expression may, if needed, tie together two or more inputs; for example, “either a credit card number, or a bank account number must be given”, or “if product id starts with ‘M’, no shipping method need be given”.

Additionally, this expression may refer to things other than inputs; for example; “if user’s credit rating is ‘excellent’ or better, Social Security number is not required”, or “if product weight is less than 1 lb., no shipping myth did need be given”.

A “computed” input is so named because it is meant to be computed automatically by some tool, by inspecting the makeup of the composite process. The language used to represent a computed input is not yet specified, and will be the subject of future work; hence, the use of Thing as range. It will require expressiveness greater than that of OWL.

Binding and Dataflows  In a composite Process the Outputs of one Process-component might be the Input of another. Moreover the outputs of a composite process may be derived from outputs of some of its components. This is a type of data flow.

Consider the above example, where a composite process is has two steps.

\[ I_1 \rightarrow \{ \text{Composite Process: CP} \} \rightarrow O_1 \]

\[ I_{11} \]

\[ \text{Step1: Perform S1} \rightarrow O_{11} \]

\[ I_{12} \]

\[ I_{21} \rightarrow \{ \text{Step2: Perform S2} \} \rightarrow O_{21} \]

Also the above equalities take place

\[ I_{11} (\text{Step1}) = I_1(\text{CP}) + 1 \]
\[ I_{12} (\text{Step1}) = \text{"Academic"} \]
\[ I_{21} (\text{Step2}) = O_{11}(\text{Step1}) \]
\[ O_1 (\text{CP}) = P_1 \ast \text{max}(0, O_{21}(\text{Step2})) \]

Each of these equalities is represented in OWL-S as a Binding, an abstract object with two properties: toParam, the name of the parameter (e.g. \( I_{21}(\text{S2}) \)), and valueSpecifier, a description of its value. In an effort to provide value specifications in as concise a manner as possible in a variety of situations, we provide four different types:

1. valueSource, the simplest sort of dataflow specification, subProperty of valueSpecifier.
_notice that the range of fromProcess is Perform. According to OWL-S, that is because “it makes no sense to refer to a value extracted from a process, because every time the process is invoked different values will be involved”.

Thus in order to express the “current value” of a parameter, that is, its value during an actual Perform, a standard variable is introduced to play the role of P, and it has been given the name TheParentPerform.

We want to say: During any Perform P of CP, the value of the input I1 of step S1 is the value of the input I1 of P.

Thus, in the above example, the input I11 equals the I1 of the overall process CP plus 1. We cannot actually refer to a Binding with valueSource that is a ValueOf with fromProcess = CP, because CP is not a Perform, but a Process. This an example where TheParentPerform is used. Also, Produce is a class used to capture dataflows to the outputs of the ParentPerform. The outputs can’t be declared once and for all, because in the presence of IfThenElses those outputs will depend on which branch of the conditional the agent takes. So at the point in a branch where the data required to compute the output are known, OWL-S inserts a Produce pseudo-step to say what the output will be.

2. valueType, is a URI referring to an OWL class definition C. An instance of valueType asserts that the value of the given parameter will belong to C. C must be a subclass of the parameter’s overall type, declared using parameterType.
3. `valueData`, is an XML literal to be interpreted as constant data

4. `valueFunction`, is an XML literal to be read as a functional term. Some of its subterms may be `ValueOfs` specifying outputs of previous terms. As with conditions and effects, the denotation of the `valueFunction` expression cannot be known until variable values are plugged in.

If we want to express the above equalities we can write something like

```xml
<process:CompositeProcess rdf:ID="CP1">
  <process:hasInput rdf:ID="I1"/>
  <process:hasOutput rdf:ID="O1"/>
  <process:composedOf>
    <process:Sequence rdf:ID="CP1">
      <process:components rdf:parseType="Collection">
        <process:Perform rdf:ID="Step1">
          <process:process rdf:resource="& aux ;# S1"/>
          <process:hasDataFrom>
            <process:InputBinding>
              -- I11 = I1(CP) + 1 --
              <process:toParam rdf:resource="& aux ;# I11"/>
              <process:valueFunction expressionLanguage="& drs ;" rdf:parseType="Literal">
                <drs:Functional_term>
                  <drs:term_function rdf:resource="& arith ;# incr"/>
                  <drs:term_args rdf:parseType="Collection">
                    <process:valueOf>
                      <process:theVar rdf:resource="#I1"/>
                      <process:fromProcess rdf:resource="# TheParentPerform"/>
                    </process:valueOf>
                  </drs:term_args>
                </drs:Functional_term>
              </process:valueFunction>
            </process:InputBinding>
          </process:hasDataFrom>
        </process:Perform>
        <process:Perform rdf:ID="Step2">
          <process:process rdf:resource="& aux ;# S2"/>
          <process:hasDataFrom>
            <process:InputBinding>
              -- I12 = "Academic" --
              <process:toParam rdf:resource="& aux ;# I12"/>
              <process:valueData xsd:datatype="& xsd ;# string">
                "Academic"
              </process:valueData>
            </process:InputBinding>
          </process:hasDataFrom>
        </process:Perform>
      </process:components>
    </process:Sequence>
  </process:composedOf>
</process:CompositeProcess>
```
<! -- I21 = O11 -->
  <process:toParam rdf:resource="&aux;#I21"/>
  <process:valueSource>
    <process:ValueOf>
      <process:theVar rdf:resource="&011;"/>
      <process:fromProcess rdf:resource="&Step1;"/>
      </process:ValueOf>
    </process:valueSource>
  </process:hasDataFrom>
</process:Perform>
</process:components>

<process:Produce> <!-- a pseudo step for the ParentPerform -->
  <process:producedBinding>
    <process:OutputBinding>
      <!-- O1 = Pi * max(0, O21) -->
      <process:toParam rdf:resource="&O1;"/>
      <process:valueFunction expressionLanguage="&drs;"
        rdf:parseType="Literal">
        <drs:Functional_term>
          <drs:term_function rdf:resource="&arith;#times"/>
          <drs:term_args rdf:parseType="Collection">
            <xsd:Integer rdf:datatype="&xsd;#Float">
              3.14159</xsd:Float>
          </drs:term_args>
          <drs:Functional_term>
            <drs:term_function rdf:resource="&arith;#max"/>
            <drs:term_args rdf:parseType="Collection">
              <xsd:Integer rdf:datatype="&xsd;#Integer">
                0</xsd:Integer>
            </drs:term_args>
          </drs:Functional_term>
          <drs:Functional_term>
            <drs:term_function rdf:resource="&arith;#max"/>
            <drs:term_args rdf:parseType="Collection">
              <xsd:Integer rdf:datatype="&xsd;#Integer">
                0</xsd:Integer>
            </drs:term_args>
          </drs:Functional_term>
        </drs:term_function>
      </process:valueOf>
      <process:toParam rdf:resource="&021;"/>
      <process:fromProcess rdf:resource="&S2;"/>
    </process:valueOf>
  </process:OutputBinding>
</process:producedBinding>
</process:Sequence>
</process:composedOf>
</process:CompositeProcess>
4.1.7 Find Cheaper Book Process Model

The Find Cheaper Book (Figures 11 and 12) process has the following sequence of processes.

Figure 11: Find Cheaper Book – Composite Process

The first process (Perform) is the BookFinderProcess which takes the book name (string) as input from the user and returns the information of a book whose title best matches the given string. Then two processes concurrently (split-join) take that output as their input, AmazonPriceProcess and BNPriceProcess, and return the book price (if any) of each bookstore respectively. Finally, these two prices are inputs in the ComparePriceProcess where they are compared and the smaller price is exported.

4.1.8 Grounding

The Profile and Process Model are considered to be abstract specifications, in the sense that they do not specify the details of particular message formats, protocols, and network addresses by which a Web service is instantiated [12]. Grounding provides a concrete specification of the required for interaction elements, i.e. the input and output messaged and atomic processes.

For describing Grounding Web Service Description Language (WSDL). The concept of Grounding is consistent with WSDL’s binding. Meanwhile some extensions for WSDL are proposed.

3Screen shots and diagram exported by the OWL-S Editor plugin in Protege [5].
Figure 12: Find Cheaper Book – Compare Prices

**OWL-S and WSDL Mappings**  
OWL-S and WSDL do not cover the same conceptual space: WSDL is unable to describe the semantics of an OWS class, and OWL-S has no means to express the binding information. But WSDL’s “abstract types” overlap with OWL-S’ inputs and outputs (Figure 13).

**OWL-S Grounding Class**  
A WsdlGrounding is just a collection of WsdlAtomicProcessGrounding instances, one for each atomic process in the process model (Figure 14).

WsdlAtomicProcessGrounding relates elements of a OWL-S atomic process to a WSDL specification. Each instance of WsdlAtomicProcessGrounding must have exactly one value for owlsProcess and exactly one for wsdlOperation.

WsdlOperationRef provides a unique specification of a WSDL operation.

XSLT Transformation transforms OWL-S parameters (inputs and outputs) to and from WSDL messages.

WsdlInputMessageMap shows how to derive a WSDL message part from (one or more) inputs of an atomic process. A WsdlInputMessageMap must have one wsdlMessagePart, and either one owlsParameter or one xsltTransformation.

WsdlOutputMessageMap shows how to derive an atomic process output from the input message parts of a WSDL operation. A WsdlOutputMessageMap must have one owlsParameter, and either one wsdlMessagePart or one xsltTransformation.

### 4.1.9 Some criticism related to Learning Services

During this research we will need to construct an ontology to describe a specific kind of Services, that of Learning Services.

To our view, in a Learning Domain discrete roles negotiate in order to achieve their goals. Also learning products (i.e Learning Objects) are not static. They can automatically assembly new custom objects and adapt to Learner’s needs during the learning process. In short,

- OWL-S does not define goals and capabilities of Learning Services separately, it models them as non functional properties in the Profile.
Figure 13: Mapping between OWL-S and WSDL

Figure 14: OWL-S Grounding
• dynamic services: products in OWL-S are static. In our case learning objects are composable and adaptive. Planning issues should be implemented. The defect for us is that each process in OWL-S is a pre-compiled plan.

• Monitoring of a Process is not implemented. For our purposes we can use KOrder [10], [11]

4.2 WSMO

Web Service Modeling Ontology is an ontology currently developed to support the deployment and interoperability of Semantic Web Services [14], [6].

It is consisted of four main elements for describing such services (Figure 15):

• Ontologies, that provide the terminology used by other elements,

• Goals, that define the problems that should be solved by Web Services,

• Web Services descriptions, that define various aspects of a Web Service, and

• Mediators, which bypass interoperability problems

5 Tools

Here lie some software tools that we will use during this research.

5.1 Protege

Protege is a free, open source platform that provides a set of tools to construct, visualize, and manipulate ontologies. It is written in Java. Protg has two main ways of modeling ontologies: (a) the Protege-Frames editor, which enables users to build and populate ontologies that are frame-based, in accordance with the Open Knowledge Base Connectivity protocol (OKBC), and (b) Protg-OWL editor, which enables users to build ontologies for the Semantic Web, in particular in the W3C’s Web Ontology Language (OWL).
Also for the purpose of this research we use Protege\textsuperscript{4} plugins: owl-s editor\textsuperscript{5}, Ontoviz\textsuperscript{6}, and Owlviz\textsuperscript{7} (Also graphviz\textsuperscript{8} should be installed).

5.2 Some Conclusions and future work

In this report we discussed about Semantic Web Services and frameworks that provides the conceptual and technical means to realize such services: mainly OWL-S and some reference on WSMO.

We described the basic components of OWL-S (profile, process and grounding), while demonstrating an example (Book Finder).

Our hint is that OWL-S can provide appropriate semantic descriptions of services but still cannot determine its applicability in our context.

Further research need to be done on Semantic Learning Services and its processes i.e. the Learning Objects.

\textsuperscript{4}http://protege.stanford.edu/
\textsuperscript{5}http://owlseditor.semwebcentral.org/download.shtml
\textsuperscript{6}http://protegewiki.stanford.edu/wiki/OntoViz
\textsuperscript{7}http://www.co-ode.org/downloads/owlviz/
\textsuperscript{8}http://www.graphviz.org/
References


A  Acronyms and Abbreviations

Some Acronyms and Abbreviations used in this report.

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LO</td>
<td>Learning Object</td>
</tr>
<tr>
<td>LOM</td>
<td>Learning Object Metadata</td>
</tr>
<tr>
<td>RDF</td>
<td>Resource Description Framework</td>
</tr>
<tr>
<td>URI</td>
<td>Universal Resource Identifier</td>
</tr>
<tr>
<td>XML</td>
<td>Extensible Markup Language</td>
</tr>
<tr>
<td>SWS</td>
<td>Semantic Web Services</td>
</tr>
<tr>
<td>WSDL</td>
<td>Web Service Description Language</td>
</tr>
<tr>
<td>SOAP</td>
<td>Simple Object Access Protocol</td>
</tr>
<tr>
<td>UDDI</td>
<td>Universal Description, Discovery and Integration</td>
</tr>
<tr>
<td>WSMO</td>
<td>Web Service Modeling Ontology</td>
</tr>
<tr>
<td>SCORM</td>
<td>Sharable Content Object Reference Model</td>
</tr>
<tr>
<td>SCO</td>
<td>Shareable Content Object</td>
</tr>
<tr>
<td>WBT</td>
<td>Web Based Training</td>
</tr>
<tr>
<td>CBT</td>
<td>Computer Based Training</td>
</tr>
<tr>
<td>IMS</td>
<td>Instructional Management Systems</td>
</tr>
<tr>
<td>[GLC]</td>
<td>Global Learning Consortium</td>
</tr>
<tr>
<td>PAPI</td>
<td>Public And Private Information</td>
</tr>
<tr>
<td>LIP</td>
<td>Learner Information Package</td>
</tr>
<tr>
<td>CROP</td>
<td>(Knowledge) Concept, Resource, Object, Product</td>
</tr>
<tr>
<td>SHOE</td>
<td>Simple HTML ontology extensions</td>
</tr>
</tbody>
</table>

B  WSDL document Example

The following example shows the WSDL definition of a simple service that adds two numbers. The service supports a single operation called addition, which is deployed using the SOAP protocol over HTTP. The service requests two integers and returns an integer, the result of their addition.

```xml
1  <wsdl:types>
2   <element name="addition">
3     <complexType>
4       <sequence>
5         <element name="number_a" type="xsd:int"/>
6         <element name="number_b" type="xsd:int"/>
7       </sequence>
8     </complexType>
9   </element>
10  <element name="additionResponse">
11     <complexType>
12       <sequence>
13         <element name="result" type="xsd:int"/>
14       </sequence>
15     </complexType>
16  </wsdl:types>
```
C OWL-S Expression Examples

Bellow we display two examples of Expressions in OWL-S. The first one is in KIF (Quoted Expression) and the second is in DRS (Unquoted Expression).

They both express the same thing: in order to send the number of a certain credit card (\( ?cc \)) [\( ?cc \) in KIF’s and \#CC in DRS’ example] to a web agent, one must know this number (\( ?num \)) [\( ?num \) in KIF’s and \#Num in DRS’ example].

A KIF Example
A DRS Example

Where:

?cc and #CC stand for an Input Parameter to the Process, supplied by the Client.

?num and #Num stand for a Local Parameter to the Process, set during the Process.
D OWL-S Process Example

In the next example, a process charges a credit card. The charge goes through if the card is not overdrawn. If it is overdrawn, the only output is a failure notification. So the description of the process must include the description of two Results, possibly in this form:

```xml
1 <process:AtomicProcess rdf:ID="Purchase">
  2 <process:hasInput>
  3   <process:Input rdf:ID="ObjectPurchased"/>
  4 </process:hasInput>
  5 <process:hasInput>
  6   <process:Input rdf:ID="PurchaseAmt"/>
  7 </process:hasInput>
  8 <process:hasInput>
  9   <process:Input rdf:ID="CreditCard"/>
 10 </process:hasInput>
 11 <process:hasOutput>
 12   <process:Output rdf:ID="ConfirmationNum"/>
 13 </process:hasOutput>
 14 <process:hasResult>
 15   <process:Result>
 16     <process:hasResultVar>
 17       <process:ResultVar rdf:ID="CreditLimH"/>
 18     </process:ResultVar>
 19   </process:hasResultVar>
 20   <process:inCondition>
 21     <expr:KIF-Condition>
 22       <expr:expressionBody>
 23         (and (current-value (credit-limit ?CreditCard) ?CreditLimH)
 24           (> ?CreditLimH ?purchaseAmt))
 25     </expr:expressionBody>
 26   </expr:KIF-Condition>
 27 </process:inCondition>
 28 <process:withOutput>
 29   <process:OutputBinding>
 30     <process:toParam rdf:resource="# ConfirmationNum"/>
 31     <process:valueFunction rdf:parseType="Literal">
 32       <cc:ConfirmationNum xsd:datatype="&xsd;#string"/>
 33     </process:valueFunction>
 34   </process:OutputBinding>
 35 </process:withOutput>
 36 <process:hasEffect>
 37   <expr:KIF-Condition>
 38     <expr:expressionBody>
 39       (and (confirmed (purchase ?purchaseAmt) ?ConfirmationNum)
 40         (own ?objectPurchased)
 41         (decrease (credit-limit ?CreditCard))
```
?purchaseAmt))
  </expr:exprBody>
</expr:KIF-Condition>
</process:hasEffect>
</process:Result>
<process:Result>
  <process:hasResultVar>
    <process:ResultVar rdf:ID="CreditLimL">
      <process:parameterType rdf:resource="&ecom;#Dollars"/>
    </process:ResultVar>
  </process:hasResultVar>
  <process:inCondition>
    <expr:KIF-Condition>
      <expr:exprBody>
        (and (current-value (credit-limit ?CreditCard)
        ?CreditLimL)
        (< ?CreditLimL ?purchaseAmt))
      </expr:exprBody>
    </expr:KIF-Condition>
  </process:inCondition>
  <process:withOutput rdf:resource="&ecom; failureNotice"/>
  <process:OutputBinding>
    <process:toParam rdf:resource="# ConfirmationNum"/>
    <process:valueData rdf:parseType="Literal">
      <drs:Literal>
        <drs:litdefn xsd:datatype="&xsd;#string"
          >00000000</drs:litdefn>
      </drs:Literal>
    </process:valueData>
  </process:OutputBinding>
</process:withOutput>
</process:Result>
</process:hasResult>
</process:AtomicProcess>