An OWL-S Editor Tutorial

Version 1.1

Shahin Saadati and Grit Denker

SRI International

Menlo Park, CA 94025
Abstract

The power of Web Service (WS) technology lies in the fact that it establishes a common, vendor-neutral platform for integrating distributed computing applications, in intranets as well as the Internet at large. Semantic Web Services (SWSs) promise to provide solutions to the challenges associated with automated discovery, dynamic composition, enactment, and other tasks associated with managing and using service-based systems. One of the barriers to a wider adoption of SWS technology is the lack of tools for creating SWS specifications. OWL-S is one of the major SWS description languages. This tutorial presents a practical guide to creating OWL-S ontologies using the OWL-S Editor. The OWL-S Editor is a tool whose objective is to allow easy, intuitive OWL-S service development for users who are not experts in OWL-S and to provide a variety of special-purpose capabilities to facilitate SWS design. The editor is implemented as a plugin to the Protégé OWL ontology editor. The OWL-S Editor is developed as open-source software and can be downloaded from http://owlseditor.semwebcentral.org/.
**Table of Figures**

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Protege folder after installation of OWL-S Editor</td>
<td>7</td>
</tr>
<tr>
<td>2</td>
<td>Creating a new project in Protégé</td>
<td>8</td>
</tr>
<tr>
<td>3</td>
<td>Creating a new project</td>
<td>9</td>
</tr>
<tr>
<td>4</td>
<td>Enabling the OWL-S Editor tab</td>
<td>10</td>
</tr>
<tr>
<td>5</td>
<td>Dialog box for reloading a project to make changes permanent</td>
<td>10</td>
</tr>
<tr>
<td>6</td>
<td>Choosing project or file names</td>
<td>11</td>
</tr>
<tr>
<td>7</td>
<td>Choosing project directory and name</td>
<td>12</td>
</tr>
<tr>
<td>8</td>
<td>Choosing project and OWL file names for the BravoAir service</td>
<td>12</td>
</tr>
<tr>
<td>9</td>
<td>The OWL-S Editor tab in Protege</td>
<td>13</td>
</tr>
<tr>
<td>10</td>
<td>Namespace that are declared when OWL-S Editor tab is enabled</td>
<td>13</td>
</tr>
<tr>
<td>11</td>
<td>Imported ontologies when OWL-S Editor tab is enabled</td>
<td>13</td>
</tr>
<tr>
<td>12</td>
<td>Instance panes and tool bar of the OWL-S Editor tab</td>
<td>16</td>
</tr>
<tr>
<td>13</td>
<td>Top-level OWL-S classes and their relationships</td>
<td>17</td>
</tr>
<tr>
<td>14</td>
<td>Inputs and outputs of SearchFlight atomic process</td>
<td>18</td>
</tr>
<tr>
<td>15</td>
<td>Inputs and outputs of SelectFlight atomic process</td>
<td>18</td>
</tr>
<tr>
<td>16</td>
<td>Inputs and outputs of MakeReservation atomic process</td>
<td>18</td>
</tr>
<tr>
<td>17</td>
<td>Defining classes of the service domain ontology</td>
<td>19</td>
</tr>
<tr>
<td>18</td>
<td>Dialog for choosing a process class</td>
<td>20</td>
</tr>
<tr>
<td>19</td>
<td>Naming the process instance</td>
<td>20</td>
</tr>
<tr>
<td>20</td>
<td>Parameter fields in the individual editor of an atomic process</td>
<td>21</td>
</tr>
<tr>
<td>21</td>
<td>Selecting the type of class on an input parameter</td>
<td>22</td>
</tr>
<tr>
<td>22</td>
<td>Choosing a parameter type</td>
<td>23</td>
</tr>
<tr>
<td>23</td>
<td>Specifying name and parameter type of an input</td>
<td>24</td>
</tr>
<tr>
<td>24</td>
<td>Parameters of SearchFlight process</td>
<td>25</td>
</tr>
<tr>
<td>25</td>
<td>Naming a composite process in the individual editor under the &quot;Properties&quot; tab</td>
<td>26</td>
</tr>
<tr>
<td>26</td>
<td>The Visual Editor for composite processes</td>
<td>27</td>
</tr>
<tr>
<td>27</td>
<td>Button to create control constructs</td>
<td>27</td>
</tr>
<tr>
<td>28</td>
<td>Empty sequence</td>
<td>28</td>
</tr>
<tr>
<td>29</td>
<td>Sequence with an empty perform</td>
<td>29</td>
</tr>
<tr>
<td>30</td>
<td>Assigning a process to a perform statement</td>
<td>30</td>
</tr>
<tr>
<td>31</td>
<td>Chooser for the process type</td>
<td>31</td>
</tr>
<tr>
<td>32</td>
<td>Individual editor for a perform</td>
<td>32</td>
</tr>
<tr>
<td>33</td>
<td>Perform in tree and process graph</td>
<td>33</td>
</tr>
<tr>
<td>34</td>
<td>BravoAirReservation composite process</td>
<td>33</td>
</tr>
<tr>
<td>35</td>
<td>Abbreviated names for perform nodes in tree view</td>
<td>34</td>
</tr>
<tr>
<td>36</td>
<td>Full names for perform nodes in tree view</td>
<td>34</td>
</tr>
<tr>
<td>37</td>
<td>Highlighting nodes in the tree view and the process graph</td>
<td>35</td>
</tr>
<tr>
<td>38</td>
<td>Process graph menu for zooming, printing, and exporting</td>
<td>36</td>
</tr>
<tr>
<td>39</td>
<td>Parameter binding widget</td>
<td>37</td>
</tr>
<tr>
<td>40</td>
<td>Selecting a Perform statement for binding a parameter</td>
<td>38</td>
</tr>
<tr>
<td>41</td>
<td>Selecting the parameter in the Perform statement that will be bound</td>
<td>38</td>
</tr>
<tr>
<td>42</td>
<td>Binding for the parameter FlightIteneraries</td>
<td>39</td>
</tr>
<tr>
<td>43</td>
<td>Control and data flow in the process graph</td>
<td>40</td>
</tr>
<tr>
<td>44</td>
<td>BravoAir grounding instance</td>
<td>41</td>
</tr>
</tbody>
</table>
Figure 45 Missing service for the BravoAir grounding  41
Figure 46 Chooser dialog for WSDL atomic process grounding  42
Figure 47 Properties of a WSDL atomic process grounding  43
Figure 48 Dialog to choose atomic process for grounding  44
Figure 49 Grounding for atomic process SearchFlight  44
Figure 50 Atomic process groundings for all atomic process of the composite process  
  BravoAir  45
Figure 51 BravoAir WSDL document location  45
Figure 52 WSDL widget  47
Figure 53 OWL-S option for WSDL input map  48
Figure 54 XSLT String option for WSDL input map  48
Figure 55 Creating WSDL message parts  48
Figure 56 BravoAir WSDL message parts  49
Figure 57 OWL-S parameter chooser for output message maps  50
Figure 58 BravoSearchFlightGrounding WSDL message maps  51
Figure 59 WSDL operation for BravoAirFlightGrounding  52
Figure 60 Creating the BravoAir service profile  53
Figure 61 Input, output, parameters of BravoAir profile  54
Figure 62 BravoAir service relations to profile, process, and grounding  55
Figure 63 IOPR Manager  56
Figure 64 Options window  58
Figure 65 WSDL2OWLs dialog  59
Figure 66 Graph Overview of BravoAir Example  60
Figure 67 Service execution  60
Figure 68 Interface for input specification and output display of service execution  61
1. Introduction

OWL-S is an ontology of service concepts that supplies a Web service designer with a core set of markup language constructs for describing the properties and capabilities of a Web service in unambiguous, computer-interpretable form. Following the layered approach to markup language development, the current version of OWL-S builds on the Ontology Web Language (OWL) Candidate Recommendation produced by the Web-Ontology Working Group at the World Wide Web Consortium. OWL-S introduces ontologies to describe, on the one hand, the concepts in the services' domain (e.g., flights and hotels, tourism, e-business), and on the other hand, generic concepts to describe the services themselves (e.g., control flow, data flow) and how they relate to the domain ontologies (via inputs and outputs, preconditions and effects, and so on). These semantically rich descriptions enable automated machine reasoning over service and domain descriptions, thus supporting automation of service discovery, composition, and execution, and reducing manual configuration and programming efforts.

This is a tutorial about the OWL-S Editor that is implemented as a Protégé plugin. The reader must be familiar with both Protégé\(^1\) and OWL\(^2\). Some knowledge on Web services and how they work is also necessary, though we present a brief introduction to OWL-S in the following section.

---

\(^1\) [http://protege.stanford.edu](http://protege.stanford.edu)

\(^2\) [http://protege.stanford.edu/plugins/owl](http://protege.stanford.edu/plugins/owl)
2. Brief Overview of OWL-S

OWL-S organizes a service description into four conceptual areas: the *process model*, the *profile*, the *grounding*, and the *service*.

A process model describes how a service performs its tasks. It includes information about inputs, outputs (including a specification of the conditions under which various outputs will occur), preconditions (circumstances that must hold before a service can be used), and results (changes brought about by a service). The process model differentiates between composite, atomic, and simple processes. For a composite process, the process model shows how it breaks down into simpler component processes, and the flow of control and data between them. Atomic processes are essentially "black boxes" of functionality, and simple processes are abstract process descriptions that can relate to other composite or atomic processes.

A profile provides a general description of a WS, intended to be published and shared to facilitate service discovery. Profiles can include both functional properties (inputs, outputs, preconditions, and results) and nonfunctional properties (service name, text description, contact information, service category, and additional service parameters). The functional properties are derived from the process model, but it is not necessary to include all the functional properties from the process model in a profile. A simplified view can be provided for service discovery, on the assumption that the service consumer would eventually look at the process model to achieve a full understanding of how the service works.

A grounding specifies how a service is invoked, by detailing how the atomic processes in a service’s process model map onto a concrete messaging protocol. OWL-S provides for different types of groundings to be used, but the only type developed to date is the WSDL grounding, which allows any WS to be marked up as an SWS using OWL-S.

A service simply binds the other parts together into a unit that can be published and invoked. It is important to understand that the different parts of a service can be reused and connected in various ways. For example, a service provider may connect its process model with several profiles in order to provide customized advertisements to different communities of service consumers. A different service provider, providing a similar service, may reuse the same process model, possibly as part of a larger composite process, and connect it to a different grounding. The relationships between service components are modeled using properties such as presents (Service-to-Profile), describedBy (Service-to-Process Model), and supports (Service-to-Grounding).
3. Installing and Running the OWL-S Editor

Before you install the OWL-S plugin, you need to install the following software.

- Java Virtual Machine (JVM).
- Protégé: Download it from http://protege.stanford.edu/download.html. Get version 3.0 or later. In the following, we assume that you installed Protégé under <Protégé-dir>.
- OWL plugin: Get it from http://protege.stanford.edu/plugins/owl/download.html. Unzip the latest binary into the <Protégé-dir>/plugins, the plugins directory of Protégé.
- Graphviz: Get it from http://www.research.att.com/sw/tools/graphviz/download.html. This component is not mandatory for running the OWL-S editor in general, but it is used for graphically displaying composite processes, and thus, it is highly recommended to install this to make full use of the OWL-S Editor capabilities. The default assumption is that Graphviz is installed under C:\Program Files\ATT\Graphviz, but one can change the location in the options button of the OWL-S Editor. More on this in Section 6.

After installing the required software you need to download the latest OWL-S plugin zip file from http://projects.semwebcentral.org/frs/?group_id=25 (or go to http://owlseditor.semwebcentral.org/ and click on download and the link for the OWL-S Editor), and unzip it under the plugins directory of Protégé, that is <Protégé-dir>/plugins. The following picture shows how the Protégé directory looks like after completing the installation steps for Protege_3.0. There may be extra directories under plugins.

![Figure 1 Protege folder after installation of OWL-S Editor](image)

That is all for installation. Now you are ready to run the OWL-S Editor. For this, you must first start Protégé by clicking on the “launch anywhere” button. When Protégé starts, it will ask you whether you want to create a new project or open an existing project.
If you choose to create a new project, then you have the choice between creating a project from an existing source (such as the owl file with the abbreviated BravoAir service description that comes along with this tutorial), or creating a new project of a specific format. In either case, you should use the “OWL Files (.owl or .rdf)” format option. The following figure shows the dialog when you choose to create a new project.

Figure 2 Creating a new project in Protégé
At this point, the OWL-S plugin is disabled. To enable it, you need to go to the Configure submenu under the Project menu and check the Owlstab as shown below.
In order to have this change take effect, you must reload the project.

If you click on the “Yes” button, you will be asked to provide a name for the new project that you are currently working on. You have the choice between specifying a new name
by just typing it into the “Project” field, or to browse through existing folders by clicking on the button. In the latter case, you can either select an existing project, or browse through the directory where you want to save your new project and give it a new name. Browsing through existing folders has the advantage that the path to the file is automatically spelled correctly.

![Image of OWL Files dialog box](image)

**Figure 6 Choosing project or file names**

For our example, we chose to browse through the directory until we found Protégé’s example subdirectory by clicking on the button. Clicking on the *examples* directory, we specified the new name “BATutorial” for our project. Protégé automatically provides the .pprj extension of the file name.
The resulting name for the newly created project is therefore “BATutorial.pprj” and the OWL part of our project will be written into the file “BATutorial.owl” (in the same directory in which BATutorial.pprj is located).
Once you click okay, Protégé will save the new project. Since all this was triggered by the fact that we wanted to enable the OWL-S Editor tab, after saving the project, Protégé will show the OWL-S Editor tab next to the other Protégé tabs.

![Figure 9 The OWL-S Editor tab in Protege](image)

In addition, for several ontologies namespace declarations have been defined or ontologies have been imported. This is the result of the enabling the OWL-S Editor tab. You can see the imported ontologies by clicking on the Metadata tab.

<table>
<thead>
<tr>
<th>Prefix</th>
<th>Namespace</th>
</tr>
</thead>
<tbody>
<tr>
<td>rdfs</td>
<td><a href="http://www.w3.org/2000/01/rdf-schema#">http://www.w3.org/2000/01/rdf-schema#</a></td>
</tr>
<tr>
<td>owl</td>
<td><a href="http://www.w3.org/2002/07/owl#">http://www.w3.org/2002/07/owl#</a></td>
</tr>
<tr>
<td>xsd</td>
<td><a href="http://www.w3.org/2001/XMLSchema#">http://www.w3.org/2001/XMLSchema#</a></td>
</tr>
<tr>
<td>service</td>
<td><a href="http://www.daml.org/services/owl-s/1.1/Service.owl#">http://www.daml.org/services/owl-s/1.1/Service.owl#</a></td>
</tr>
<tr>
<td>expr</td>
<td><a href="http://www.daml.org/services/owl-s/1.1/generic/Expression.owl#">http://www.daml.org/services/owl-s/1.1/generic/Expression.owl#</a></td>
</tr>
<tr>
<td>cont</td>
<td><a href="http://www.isi.edu/~pan/damltime/timezone-ont.owl#">http://www.isi.edu/~pan/damltime/timezone-ont.owl#</a></td>
</tr>
<tr>
<td>process</td>
<td><a href="http://www.daml.org/services/owl-s/1.1/Process.owl#">http://www.daml.org/services/owl-s/1.1/Process.owl#</a></td>
</tr>
<tr>
<td>grounding</td>
<td><a href="http://www.daml.org/services/owl-s/1.1/Grounding.owl#">http://www.daml.org/services/owl-s/1.1/Grounding.owl#</a></td>
</tr>
<tr>
<td>ist</td>
<td><a href="http://www.daml.org/services/owl-s/1.1/generic/ObjecList.owl#">http://www.daml.org/services/owl-s/1.1/generic/ObjecList.owl#</a></td>
</tr>
<tr>
<td>swrl</td>
<td><a href="http://www.w3c.org/2003/01/swrl#">http://www.w3c.org/2003/01/swrl#</a></td>
</tr>
<tr>
<td>swrlb</td>
<td><a href="http://www.w3.org/2003/01/swrlb#">http://www.w3.org/2003/01/swrlb#</a></td>
</tr>
<tr>
<td>profile</td>
<td><a href="http://www.daml.org/services/owl-s/1.1/Profile.owl#">http://www.daml.org/services/owl-s/1.1/Profile.owl#</a></td>
</tr>
<tr>
<td>time</td>
<td><a href="http://www.isi.edu/~pan/damltime/time-entry.owl#">http://www.isi.edu/~pan/damltime/time-entry.owl#</a></td>
</tr>
</tbody>
</table>

![Figure 10 Namespace that are declared when OWL-S Editor tab is enabled](image)

If you click on the OWL-S Editor tab, you will get a window that provides a more direct view of OWL-S classes and instances than what Protégé provides by default. The OWL-S tab is separated into two parts. The left-hand side provides the so-called instance panes
which allow users to easily navigate their service descriptions. The instance panes list all
instances of a service, divided into Service, Profile, Process, and Grounding instances.
The right-hand side of the OWL-S tab is an editing pane, also called Individual Editor,
that changes depending on the selection in the instance panes, to show a specialized
editing mode for the chosen type of OWL-S instance. For example, if the user selects a
profile instance (used for service discovery), then the right pane will show all properties
of the profile, allowing the user to create fine-tuned service advertisements. If a
composite process is selected, the editing pane changes to a graphical process editor (see
Section Composite Process).

There is also an OWL-S-specific toolbar on the top left of the OWL-S Editor tab. It
shows the following icons . The first icon will activate the
Input/Output/Precondition/Result Manager, also called IOPR Manager, an integrated
facility to manage IOPRs and their association with processes and profiles. This feature is
described in Section IOPR Manager. The second icon activates a program to generate
template OWL-S services from existing WSDL description and will be described in detail
in Section Importing WSDL Files. The third icon will enable a graphical overview of the
relations between services, profiles, processes, and groundings of the services at hand.
More details are given in Section Graph Overview. Finally, the last icon provides
additional options, such as specifying a path to the GraphViz software, and is explained
in more detail in Section Options.
The above screenshot does not show any instances for any of the OWL-S subontologies since we started a new project. If we would have loaded an existing OWL-S project, we would see all instances listed in the appropriate instance panes. To create a new instance of either service, profile, process or grounding, you can click on the button.

In the remainder of this tutorial we will show how to create an OWL-S service description. For the rest of this tutorial it is assumed that all editing is done in the OWL-S editor tab, unless otherwise noted.

Generally, there is no need to work with the other Protégé tabs such as OWL Classes and Properties if the following condition holds: As long as the user has defined the service domain ontologies, she can define the OWL-S service by exclusively using our OWL-S Editor tab. With service domain ontologies we refer to those ontologies that define the application domain for which one wants to create a service (for example, the domain ontologies for an online book selling service will contain information about books, credit cards, and so on). If the user has not yet defined those ontologies, then she would likely use Protégé’s OWL Classes, Properties and Individuals tabs to do so.

You are now ready to create your first OWL-S Service.
4. Creating an OWL-S Service

BravoAir is an imaginary Web service which provides an online reservation and purchasing service for airplane tickets. We shall create a simplified version of the BravoAir example put forth by the OWL-S coalition in http://www.daml.org/services/owl-s/1.1/examples.html.

Our BravoAir Web service provides the following functionalities:

- Search for a flight,
- Select a desired flight
- Make a reservation

Every OWL-S Service, that is every instance of the OWL-S class Service, is related to at most one instance of the OWL-S class ServiceModel via the property describedBy. There is no restriction on how many Profiles are related to a service via the property presents. A ServiceGrounding is related to exactly one service via the property supportedBy, the inverse property to supported. The following displays the relationships between the four top-level OWL-S classes (we do not show the inverse properties).

![Diagram of OWL-S classes and their relationships]

Figure 13 Top-level OWL-S classes and their relationships

In the rest of this section, we will go step-by-step through the creation of instances for each of the four classes with the help of our BravoAir Web service.

4.1. Creating a Service Process

4.1.1. Atomic Processes

We start with creating instances of the Process class. We intend to describe three different kinds of activities for our BravoAir service, namely search for flights, select a flight, and make a reservation. We decided to model all three of these as atomic processes, as we do not intend to further refine them. The decision what is an atomic process and what is a composite process that can be further refined into a combination of atomic processes is up to the designer.
Furthermore, we need to decide what are the inputs and outputs for each of the atomic processes. The following diagrams show the inputs and outputs for our atomic processes.

**Figure 14 Inputs and outputs of SearchFlight atomic process**

- **SearchFlight**:
  - FromCity (City)
  - Date (Date)
  - NumberOfPassengers (PositiveIntegers)
  - **FlightItineraries** (List of Itinerary)

**Figure 15 Inputs and outputs of SelectFlight atomic process**

- **SelectFlight**:
  - FlightItineraries (List of Itinerary)
  - **FlightItinerary** (Itinerary)
  - DepartDate (Date)
  - DepartTime (Time)
  - Price (Decimal)

**Figure 16 Inputs and outputs of MakeReservation atomic process**

- **MakeReservation**:
  - FlightItinerary (Itinerary)
  - Address (TheAddress)
  - PersonName (Name)
  - CreditCardNumber (CreditCardNum)
  - ExpirationDate (Date)
  - Confirmed (Boolean)
  - ReservationID (Integer)

The name of each input or output is specified (in black) as well as a type is defined for each input and output (in brackets, written in blue). Inputs and outputs are subclasses of
the more general parameter class. A parameter has a type, `process:parameterType` which is of type `xsd:anyType`.

Practically, there are two cases how to assign the parameter type to a parameter. The first is to wrap a given OWL class in a URI by adding to the class name the information necessary to make it URI. The result of a wrapped OWL class is of type `xsd:anyURI`. The second possibility is to choose one of the predefined `xsd:anyTypes` such as `xsd:integer`. The OWL-S Editor supports the wrapping of OWL classes into URIs as well as choosing from a set of predefined XSD datatypes. Thus, when defining the parameter type of a parameter, the user can choose between those OWL classes she has defined previously and the predefined XSD datatypes. The OWL-S Editor does automatically generate the correct URIs.

As mentioned before, the domain ontologies must be already be defined in order to use them in the OWL-S Editor tab. Classes describing parameter types of inputs and outputs are part of these domain ontologies. Thus, we will start with defining those classes that will be used as parameter types for input or outputs of our atomic processes. The declaration of these classes is done in the `OWLClasses` tab of Protégé, not the OWL-S Editor tab, because modeling classes and properties for any kind of domain is the purpose of Protégé’s OWL plugin. The following figure shows the parameter types that we created for our BravoAir example (newly created classes have yellow icons on their left side).

![Figure 17 Defining classes of the service domain ontology](image)

Once you created all parameter types in the `OWLClasses` tab and switched back to OWL-S editor tab, you are ready to create a process instance. For this, you must click on the button provided in the process instance pane and select in the dialog box `process:AtomicProcess` as the class for which you want to create an instance.
The name of the class, such as *SearchFlight* can be typed into the name field of the individual editor that appears on the right-hand side of the OWL-S tab.

The parameter fields are part of the individual editor for the new atomic process instance.
The `hasInput` and `hasOutput` properties are the places where you can define the inputs and outputs of an atomic process. To create a new input or output, click on the button of the proper property. A new window will pop up in which you have to choose the class of the input, either an external resource for which you can define resource URI in a window that will pop up if you choose `process:ExternalResource`, or, if you choose `process:Input`, then you can define the name and the parameter type of the input in the individual editor that will pop up.
If you choose to create a new input parameter, the window that will pop up will allow you to define a name for your input parameter and requires you to specify the parameter type. You have two choices: Either you click on the icon to select one of the defined classes as parameter type, or you click on the icon to select one of the predefined XSD datatypes as parameter type.
For example, in our BravoAir example, for the atomic process *SearchFlight*, we defined an input parameter *NumberOfPassengers* and defined the parameter type of this input as *xsd:positiveInteger*. 
You must repeat this process for all inputs and outputs of your atomic process. When you finished introducing all the parameters for the search flight atomic process, your page should look like the following.
Create the other two atomic processes, SelectFlight and MakeReservation, in the same way.
4.1.2. Composite Process

At this point you have created an atomic process for each of the functionalities of our Web service. We now want to create a composite process which consists of all of the created atomic processes\(^1\). The OWL-S editor is equipped with a powerful visual tool to simplify the creation of a composite process. A composite process is constructed from subprocesses that can in turn be composite, atomic, or simple. The control flow of a composite process is defined using control constructs, such as If-Then-Else, Sequence, and Repeat-Until. These constructs can be nested to an arbitrary depth.

These control flows are particularly hard to generate by hand or in a plain ontology editor not designed for this task. We drew inspiration from workflow management tools and existing approaches to drawing process graphs. We will explain the features of the visual editor for composite process editing in the following in detail.

To create an instance of a composite process, click on the button in the process instance pane and select \texttt{process:CompositeProcess}. The \textit{Visual Editor} will automatically appear on the right side of the OWL-S tab. Note that whenever you choose to create a composite process, the default is to get the visual editor, whereas when you choose to create an atomic process, you will get the individual editor because not specific graphical support for complex control and data flow is necessary. Nevertheless, you can also see a more textual presentation of a composite process by switching to the \textit{Properties} tab.

In in the individual editor of the \textit{Properties} tab you will see process properties such as \textit{Name}, \textit{process:hasInput}, and so on. For the purpose of our example, let us call the composite process \textit{BravoFlightReservation}.

![Figure 25 Naming a composite process in the individual editor under the "Properties" tab](image)

Switch back to the visual editor to define the composite process in more detail. Though you can also make further specifications using the \textit{Properties} tab, we strongly recommend using the visual editor as it provides special-purpose capabilities that make it much easier for you to define a composite process.

\(^1\) For more information on atomic, composite and simple processes in OWL-S see \url{http://www.daml.org/services/owl-s/1.1/overview}
The visual editor consists of two parts. The left-hand side of the visual editor is initially empty. Once you define your composite process in detail, it will show a tree-like presentation of the composite process. The right-hand side of the visual editor gives you the choice between a Process graph and a Properties editor. The process graph is a visualization of the composite process as a state-oriented workflow. Initially the process graph only shows the Start/In state and the Finish/Out state. Once you add components to the composite process, both the tree presentation to the right and the process graph to the left in the visual editor will be automatically updated.

Instead of the process graph, you can also choose the Properties tab and get a less graphical, more textual representation in the form of an individual editor for the composite that you are currently defining. You can also switch between Process graph tab and Properties tab.

The left side of the visual editor provides a list of diamond-shaped buttons for the control constructs of OWL-S.

Clicking on a diamond-shaped button creates one the following:

- **P**: Perform
OWL-S control flows can be drawn using boxes for subprocess invocation (called Performs in OWL-S), diamonds for conditional nodes (e.g., for If-Then-Else constructs), and arrows showing the flow of execution. Being able to view these workflow graphs was a high priority for us. However, allowing users to more or less "draw" the workflow by hand is neither desirable nor feasible. Instead, we take advantage of the fact that all OWL-S control flows are trees in the graph-theoretical sense. Therefore, we let the user model the control flow in a GUI tree component, with full drag-and-drop support, whereas the corresponding workflow graph is updated to reflect any changes to this tree.

We assume that our BravoAir service is a sequence of the SearchFlight, SelectFlight, and MakeReservation atomic processes. By definition, a sequence in OWL-S is a sequence of performs of other atomic or composite processes. Thus, in order to create the sequence for our example service, we first click on the sequence button, which will result in a new node in the tree as well as the process graph.

![Figure 28 Empty sequence](image)

In our case we want to first perform the SearchFlight atomic process. Thus, while the cursor highlights the Sequence node in the visual editor, we click on the P button and receive the following.
The result is a perform node under the sequence node. In the process graph view one can see a red-framed box containing a question mark. This box is framed red because we must assign which process is performed. For this we move the cursor over the box with the question mark and click on it. That will result in an individual editor of the following kind.

Note that the control flow buttons are disabled while your cursor is on the perform node. This is because one cannot nest any control construct under a perform.
Clicking on the icon for choosing an instance, you get a dialog in which you have to choose the kind of process we want to assign to the perform statement. You have the choice among atomic, simple, and composite process, as well as external resources.
This dialog window also gives information how many instances of each of those process are already defined. We chose atomic processes and got the list of all atomic process that we have defined. We can select any of the listed processes by clicking on it.

We chose SearchFlight and returned to the visual editor by clicking okay and closing the individual editor window. We give a name to the perform instance of the SearchFlight atomic process, namely SearchFlightPerform, as this will be helpful later when we define data flow between the perform statements and need to identify from which perform statement we want to bind the parameter. Moreover, the visual editor will show in the tree the name of the atomic process that is performed. Thus, there is really no need to come up with a user-defined name for each perform of a process.
Initially, the visual editor does not show the name of the perform process in the process graph, but one can achieve a redrawing of the process graph through clicking on another node in the tree (such as the sequence node).
In order to create the second perform in the sequence, you have to highlight the sequence node by clicking it. This will give enable the control buttons again. Now you can click again on the perform button and repeat the steps as above, just this time choose the SelectFlight atomic process. The new perform construct is added as the last element in the sequence. If you repeat this for all three processes that should be put in the sequence, then you get the following for the composite process *BravoAirFlightReservation*.
If you want to extend the abbreviate names in the perform statements of the tree to their full name, that is, if you want to change

![Figure 35 Abbreviated names for perform nodes in tree view](image)

...to

![Figure 36 Full names for perform nodes in tree view](image)

you can do this by clicking on a perform node and highlighting it, such that an orange frame is around the node. This might require clicking the node twice. Then you can click again, hold the mouse down and drag it to the right. This will extend the name of the process associated to the perform statement.

Note that the node which is highlighted in the tree view will also be highlighted in the process graph.
You can click on each node in the process graph to see its properties or, if you have that node for which you want to see details highlighted in the tree view, you can switch from the process graph to the properties tab.

You can also drag and drop nodes in the tree view. Just click on the node you want to move and hold down the mouse. You can then move the mouse and will see the different places in which you can try to drop the node. The OWL-S Editor will monitor which of the drag and drop movements are correct with respect to the semantics of OWL-S composite processes and disallow those that are syntactically incorrect.

Also note that we support facilities such as zooming, printing and exporting the process graph to SVG in the process graph. You can enable the menu by pointing the cursor in the process graph and do a right-click.
4.1.3. Data Flow Definition

In the case of a composite process, one can define dataflow between its sub-processes. Assume we want to define that the output `FlightItineraries` of the `SearchFlight` process will become an input to the `SelectFlight` input, namely the input that has been named also `FlightItineraries`.

Generally speaking, data flow is always defined by selecting first the destination process of the data flow (e.g., `SelectFlight`), then the input parameter of this process (e.g., `FlightItineraries`) and then connect the input parameter to the appropriate output parameter (i.e., `FlightItineraries`) of the source process (i.e., `SearchFlight`). Moreover, data flow is defined and graphically represented in the visual editor.

Thus, in order to achieve the previously described dataflow, first click on the second process node in the process graph, named `SelectFlight`. Then select the parameter you want to bind in the `ToParameter` box by clicking on the button. In our case there is only one input parameter of the `SearchFlight` process that can be chosen.
Then choose the proper other parameter that you want to be bound to the input parameter that you just chose. The widget for binding a parameter gives you a choice between valueData, valueSource, and valueFunction. The first one can be used to bind a specific value to the parameter, the second to bind parameter that is elsewhere defined, and the last can be used to define a function of values and parameters.

Figure 39 Parameter binding widget

Choose the valueSource option, and choose the SearchFlightPerform as the perform statement from which the parameter will be bound.
The *SearchFlightPerform* has only one output, namely *FlightItineraries*, which we will use to bind the input parameter of *SelectFlight*.

**Figure 40 Selecting a Perform statement for binding a parameter**

**Figure 41 Selecting the parameter in the Perform statement that will be bound**
The OWL-S Editor will automatically create a representation of the dataflow in the process graph. In order to redraw the process graph, it might be necessary to switch between nodes in the tree graph. The subgraph with yellow nodes and grayish border is the control flow while the subgraph with yellow nodes and blue borders represents data flow. You can click on each process node or control construct node (i.e., yellow nodes with grey border) to see its properties. You can also right click on the graph for zooming in and out.
Figure 43 Control and data flow in the process graph
4.2. Creating a Service Grounding

In order to create a grounding click on the button in the grounding instance pane and give it a name. For our example, we chose BravoAir_Grounding as the name.

![Figure 44 BravoAir grounding instance](image)

The service:supportedBy property of the grounding instance is frame with a red box, because the cardinality restrictions in the OWL-S ontology require that each grounding is assigned to a service. We shall create the service later.

![Figure 45 Missing service for the BravoAir grounding](image)

At this point we create an instance of WsdIAtomicProcessGrounding for each atomic process that was created before. In the individual editor to the right, click on the button for the grounding:WsdIAtomicProcessGrounding property. You have to choose between WSDL atomic process grounding and an external resource.
Choosing a WSDL atomic process grounding results in a window which as, among others, the following properties defined.
Figure 47 Properties of a WSDL atomic process grounding

The *grounding:owsProcess* property is obligatory for this instance. When you click on the button to add an existing instance, you will get a choice of all atomic processes and external resources that you have defined so far.
We chose the *SearchFlight* atomic process as the one for which we want to create a grounding and gave the resulting instance of the grounding: `WSDLAtomicProcessGrounding` the name `BravoAirSearchFlightGrounding`.

You must repeat this process for all three atomic processes.
If you double-click on the \textit{BravoSearchFlightGrounding} instance, an instance editor showing the properties of \textit{BravoSearchFlightGrounding} appears. So far you have only associate an atomic process with the \texttt{grounding:owsProcess}. Other properties, such as \texttt{grounding:wsdlReference}, \texttt{grounding:wsdlDocuments} and so on, must be defined by the user. We assumed that the WSDL document of the Web services is located at \url{http://www.bravoairticket.com/BravoAir.wsdl}.

Next, mappings for the inputs and outputs from the atomic processes to the message parts of a WSDL implementation must be defined in the grounding. The following diagram summarizes the various possibilities of relating inputs and outputs with message parts.
In the simplest case, there is a one-to-one correspondence between an OWL-S input parameter and a message part of a WSDL input message part or a one-to-one correspondence between a WSDL operation output message part and an OWL-S output parameter. These correspondences are defined in the grounding of a service through so-called `grounding:WsdlMessageMaps`. In particular, the former relation is defined as a an instance of the OWL-S class `grounding:WsdlInputMessageMap` and the latter is defined as an instance of the OWL-S class `grounding:WsdlOutputMessageMap`. Lists of those maps are attached to the properties `grounding:WsdlInputs` and `grounding:WsdlOutputs` (the property names are cut off in the following screenshot).

In either of the two one-to-one correspondences, the WSDL service accepts serialized OWL, or the ontology operates on `xsd datatypes`. Often, however, a transformation has to take place, in order to map between concepts in the ontology and complex `xsd` types on the WSDL side. For these cases, the mapping between OWL-S input and output parameters can involve XSLT transformations.

To support all three kinds of WSDL message maps, we provide a special widget, the so-called `WSDL widget`. Click on one of the atomic process groundings that you have
created already, such as *BravoSearchFlightGrounding*. The individual editor shows the following WSDL widget on the right.

![WSDL widget](image)

The top part of the widget can be used to map OWL-S parameters, XSLT strings or XSLT URIs to WSDL message parts. The lower part is used to map WSDL message parts, XSLT strings or XSLT URIs to OWL-S parameters. Depending on which Wsdl option you choose, the window will change accordingly. For example, if you choose the *OWL-S Parameter* option for the input message map, then the you can choose from the defined input parameters by clicking on the button.
If you choose the *XSLT String* option, you will get a box in which you can type the XSLT transformation string.

Similarly, the *XSLT URI* provides a text box in which you can type the URI of your transformation string.

Before you can assign message maps, you have to define the WSDL message parts. This can be done by clicking the button in the *WSDL Inputs* or *WSDL outputs* box and typing the URIs of the message parts.
Since we assumed that the WSDL document for our Web services can be found at http://www.bravoairticket.com/wsdlBravoAir.wsdl, we defined the message parts with the same URL.

![Figure 56 BravoAir WSDL message parts](image)

For the OWL-S parameter of the output message maps, you can click on the + button and select from the list of all defined output parameters. Nevertheless, if you want to map a WSDL message part to a defined OWL-S parameter, you must choose the Message Part option and type the URL of the WSDL message part.
After defining all WSDL message maps for the *BravoSearchFlightGrounding*, you will get the following.
In our example, we simply map every *owlsParameter* to a *Wsd1MessagePart*. For instance the mapping of *NumberOfPassengers* is shown above.

Finally the *wsdlOperation* is also designed by the user. To create a new reference, click on the button to create an instance and choose *grounding:WsdlOperationRef* as the concrete class. In the resulting dialog box you can define a name for your WSDL operation references, such as *SearchFlight_Operation*. You also must define the URL for the operation and the URL for the port type. Here is a screenshot of the *wsdlOperation* in our example.
Figure 59 WSDL operation for BravoAirFlightGrounding
4.3. Creating a Service Profile

You can create the profile by clicking on in the profile instance pane. Name it BravoAir_Profile and relate it to the BravoAirFlightReservation process by clicking on the button and choosing the appropriate composite process. Type also an appropriate service name and a textual description.

![Figure 60 Creating the BravoAir service profile](image)

The parameters selected for the profile must be a subset of all the parameters (inputs and output) of all processes that belong to the service. It is up to the designer to choose those parameters that will be shown in the service profile. However, in most cases, the user will select all input and output parameters from all processes with the exception of internal parameters (such as FlightItineraries). The resulting profile looks as follows.
There are some optional properties that can also be entered such as the contact information.

### 4.4. Creating a Service

The last instance that needs to be created is the service. Create a service and name it `BravoAir_ReservationAgent`. In order to link the profile, process and grounding that we created earlier, we have to assign those instances to the appropriate properties. For example, the `BravoAir_Profile` instance is assigned to the `service:presents` property by clicking on the icon of that property and choosing the profile from the list of existing profiles. Similarly, the process `BravoReservationFlight` is assigned to the `service:describedBy` property and the `BravoAir_Grounding` is assigned to the `service:supports` property.
Figure 62 BravoAir service relations to profile, process, and grounding

That is all.

It is a good idea to click through all instances in the Service, Profile, Process, and Grounding instance pane and check whether the corresponding instance editor have no red-framed properties. If they do, you need to attend to this by assigning appropriate values to properties (with the exception of properties with domain rdf:List, for those the nil:List element...).

5. Input/Output/Precondition/Result (IOPR) Manager

The IOPR (Input-Output-Precondition-Result) Manager is an integrated tool where you can view, edit, and assign the inputs, outputs, preconditions, and results of the processes and profiles. This capability will be enabled by clicking on the IOPR button, the first button in the toolbar ...

Inputs, outputs, preconditions, and results (IOPRs) are important parts of services. Both profiles and processes have a set of properties to relate them to their IOPRs: hasInput, hasOutput, hasPrecondition, and hasResult. A profile usually includes a subset of the IOPRs of the process to which it is related. For this reason, it is often convenient to compare a profile side-by-side with the related process, and have them both in view when making decisions about the values of the IOPR properties. In addition, we sometimes want to relate the IOPRs of two profiles or processes (e.g., a composite process and an associated simple process, or two processes of different services).
To support efficient management of these IOPRs, we designed the *IOPR Manager*, which visualizes IOPR relationships in a very compact way (see Figure 63). Clicking the toolbar button brings up the IOPR Manager window, which is somewhat similar to the main tab widget of the OWL-S Editor. Like the tab widget, it provides four instance panes to the left, and an editing pane to the right.

![IOPR Manager Diagram](image)

**Figure 63 IOPR Manager**

The instance panes of the IOPR Manager show all the IOPRs in the KB, and allow you to create and delete IOPRs. You can also edit IOPR properties in the editing pane. In the combo boxes at the top, you can choose two profiles and/or processes that are to be compared with regard to their IOPRs. Associated with each combo box is a column of checkboxes, one for each IOPR. You can simply check or uncheck these boxes to add or remove instances of the corresponding properties (*hasInput*, *hasOutput*, and so on).
In Figure 63 we show an example of the IOPR manager for our BravoAir service. We have already specified the inputs, outputs, preconditions and results of the service and selected *BravoAir_Profile* and top-level process *BravoAirFlightReservation*. One can see the inputs and outputs that are assigned to the profile are also assigned to the process. If one wants to change the assignments of any of those, one has to click/unclick the corresponding combo boxes in the left or right column in front of the IOPRs. In addition, the editing pane shows us that the *parameterType* of input *DepartingDate* is of type *xsd:date*.

Groundings also refer to inputs and outputs. However, groundings do not refer to preconditions or effects, and the relationship with inputs and outputs is somewhat different from that of profiles and processes. For these reasons, we chose not to include the groundings in the IOPR Manager. Instead, we implemented separate support for editing groundings (see Section 4.2).

### 6. Options

If you want to view/edit the OWL-S editor options, click on the option button, that is the last button in the toolbar 🍃 🌼 🌽 🍊.

We are currently only supporting two functionalities in the options. You can define the path to the GraphViz and defining URLs for inquiring and publishing Web services using Semantic Web service registration and matching engines.
The search facility is currently under development and not functional. In the future, we plan to integrate the OWL-S Editor with existing matchmakers and registration engines for Semantic Web services, so that users can search the Web for services that fit their needs.

7. Importing WSDL Files

We have already seen that an OWL-S description relates to a WSDL file through a grounding. In many cases, it will be desirable to create a “skeletal” OWL-S description based on a preexisting WSDL file. Parts of the OWL-S description can be generated automatically based on the inputs and outputs defined in the WSDL file. We have integrated the WSDL2OWLS code from Mindswap to the OWL-S Editor, allowing users to import from a WSDL file by clicking the second button in the OWL-S toolbar.

After clicking on the WSDL button the WSDL import window will appear. The WSDL2OWLS code comes with a set of predefined URL of WSDL files that you can choose by using the scrollbar in the Enter URL field. You can also type the URL of a WSDL file.
Figure 65 WSDL2OWLS dialog

When the file is imported you will be able to view all the defined operations on the left side. You can import one operation at a time by clicking on it. On the right side you will be able to edit and modify some of the parameter names and add or edit the namespaces. When you are done you can simply click on “Import WSDL” to import your file. The OWL-S editor will ask you for a file name to complete the import.

Note that the imported data is read-only and the user can not modify it in Protégé.

8. Graph Overview

Another capability we implemented is a visualizing tool to show the relations between different components of an OWL-S service as a graph. This graph overview can be started by clicking on the third button in the toolbar.
The graph overview of our BravoAir Tutorial example is shown above. The overview shows all services, profiles, top-level processes, groundings and atomic process groundings. This gives a good overview in which way the various elements are connected. In the future we will update this further to include subprocesses, e.g., in our example we will also show the processes MakeReservation, SearchFlight, and SelectFlight. We will also show which atomic process is connected to which atomic process grounding.

If the current project has more than one service, the graph overview will show accordingly more than one tree. In the future we intend to support simple editing functionalities in the graph overview, such as adding and deleting arrows between service elements.

9. Service Execution

An exciting feature of the OWL-S Editor is the ability to actually execute services inside the editing environment. The service instance pane has a button to execute a service.

Selecting a Service instance and clicking the 'play' button will execute that service, provided that it has a WSDL grounding which is hooked up to a real Web service. The user is presented with a window where he/she gets to choose the values of the input parameters (or create new instances for them) based on the parameter types defined in the service's process model. We support four kinds of input boxes: (1) text boxes for strings, dates, and other datatypes, (2) spinner for integers, (3) combo boxes for Booleans, and (4) choosers for class instances. The execution dialog will also display the outputs of the executed service for the given inputs.
For our example, the execution dialog shows input fields such as *ToCity*, a property of type *City*. You can click on the button and create a new instance of class *City*, or you can click on the button to add an existing instance. Once all inputs are define, you can click on the execute button and see the output displayed. For now we only support the execution of atomic processes that have an actual WSDL grounding. In the future we aim to support composite as well as atomic processes, and users will be able to take the results returned from services and add them into the Protégé knowledge base.