The OWL-S Editor – A Development Tool for Semantic Web Services

Daniel Elenius, Grit Denker, David Martin,
Fred Gilham, John Khouri, Shahin Sadaati,
and Rukman Senanayake*

SRI International, Menlo Park, California, USA
firstname.lastname@sri.com

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 paper presents an OWL-S Editor, whose objective is to allow easy, intuitive OWL-S service development 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, and is being developed as open-source software.

1 Introduction

Web Services (WS) were invented to bring a new level of integration to the computing industry and its networked communities. Ideally, service-based applications should be able to interoperate despite being developed in different programming languages, at different times, by different people, with designs based on different assumptions. Standard protocols for service interface descriptions (WSDL[1]) and service invocation (SOAP[2]), coupled with a global data format (XML[3]), were introduced to turn this vision of the Service-Oriented Architecture [1] into reality.

* Supported by the Defense Advanced Research Projects Agency through the Air Force Research Laboratory under Contract F30602-00-C-0168 to SRI, and in part by Vinnova (grant no. 2002-00907) and The Swedish Research Council (grant no. 621-2003-2991).

[1] Web Service Definition Language, http://www.w3.org/TR/wdsl

[2] Simple Object Access Protocol, http://www.w3.org/TR/soap12-part0/

[3] Extensible Markup Language, http://www.w3.org/XML

A. Gómez-Pérez and J. Euzenat (Eds.): ESWC 2005, LNCS 3532, pp. 78–92, 2005.

Web Services have met with very strong initial success, mostly in the area of integration within, and (to a lesser extent) between, businesses. An increasing number of organizations are endorsing WS technology as a standardized infrastructure for interoperation of disparate software components within the organization, fulfillment of transactions between organizations, and sharing of corporate resources with customers and partners. However, this integration has been achieved only through costly efforts in manually programming and designing these WSs. Developers spend time searching for the right services and adding adapter components between incompatible services, and the resulting applications cannot adapt dynamically to changes in their environment.

Although second-generation WS specifications are under development, such as WS-CDL[4] and BPEL4WS[5], to enhance the usability, scope, and expressiveness of WSs, there is an increasing realization that technologies from the Semantic Web (SW) [2] can also make crucial contributions to WS frameworks. Semantic Web Services (SWSs) [3] take up on this idea, introducing ontologies to describe, on the one hand, the concepts in the services' domains (e.g., flights and hotels, tourism, e-business), and on the other hand, characteristics of the services themselves (e.g., control flow, data flow) and their relationships 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. The three most prominent SWS specification approaches currently under development are OWL-S [4], WSMO[6], and SWSL[7].

The field of SWSs is still in an early stage, and adoption has been slow. A limiting factor has been the lack of tool support. The objective has been to enable machines to manipulate services, yet so far arduous human work has been necessary to create the semantic service descriptions. While tools to create and edit SW ontologies in general do exist [5], modeling SWSs requires additional functionality and developer support in order to be practically feasible.

Tools that make the SWS technology accessible to a broad audience with diverse needs are a crucial factor in the success of SWS technology. Tools are needed to facilitate tasks such as service definition and annotation, execution and monitoring, and service registration and discovery. The OWL-S Editor is aimed at providing a flexible, yet powerful editor for OWL-S service definitions. This paper describes the design and current functionality of the OWL-S Editor as well as its future directions.

The remainder of the paper is organized as follows. A brief introduction to OWL-S is presented in Section 2. Section 3 forms the main part of this paper,

[4] Web Services Choreography Description Language, `http://www.w3.org/TR/2004/WD-ws-cdl-10-20041012`

[5] `http://www-128.ibm.com/developerworks/library/ws-bpel/`

[6] `http://www.wsmo.org`

[7] `http://www.daml.org/services/swsl/`

and outlines the main features of the tool. An overview of related work is given in Section 4. Future work in this area is discussed in Section 5, and Section 6 concludes with a brief summary.

2 OWL-S Overview

OWL-S is an ontology of service concepts. 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 (see Sections 3.3 and 3.4). 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 allows for different types of groundings to be used, but the only type developed to date is the WSDL grounding (see Section 3.5), which allows any WS with a WSDL definition to be marked up as a 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 OWL-S Editor: Design and Features

There are two main tasks in the development of OWL-S services. The first task is to define the service's domain ontologies in terms of OWL classes, properties, and instances. The second task is to create an OWL-S description of the service, relating this description to the domain ontologies. An OWL-S service description consists of *instances* of OWL-S classes such as `Service`, `Process`, `Input`, and `Output`. In some cases, the OWL-S ontology is also *extended* to handle specific modelling situations.

In order to best facilitate these tasks, we built the OWL-S Editor on top of the Protégé OWL Ontology Editor [5]. Protégé allows editing of domain ontologies out-of-the-box. However, efficient development of services requires additional features. Our strategy has been to leverage the existing functionality of Protégé and to utilize Protégé's pluggable architecture to extend it where we judged it would be helpful for the SWS developer. The result is a SWS development environment where the domain ontologies are well integrated with the service descriptions.

The main user interface to the OWL-S Editor is a so-called *tab widget*. Figure 1 shows the OWL-S Editor tab, which provides service-specific design capabilities as described in the following sections.

Our design also makes it easy to extend the OWL-S ontologies. A common scenario is to create subclasses of the OWL-S `Profile` class, creating a *profile hierarchy* with profiles specific for different domains. Figure 2 shows an example of a custom profile in such a hierarchy.

In addition, building our tool on top of Protégé means that users can take advantage of the many other existing Protégé plugins, e.g. for querying and visualizing the Knowledge Base (KB), and to export the KB to different formats. These different plugins coexist gracefully, all working on the same KB (see Figure 1).

The icons in the toolbar on the top left of the OWL-S Editor tab provide parameter management, generating an OWL-S service from a WSDL specification, graphical overview, and additional options. In the following sections we discuss these and other features in more detail.

3.1 Managing the Top-Level Ontology

As explained in Section 2, a number of properties connect the different components of OWL-S services. It is very important to be able to get a good overview of these relationships when developing an OWL-S service. The OWL-S Editor tab widget provides a customized view for managing instances of the OWL-S subontologies. Along the left side of the OWL-S tab are four *instance panes* (see Figure 1), one each for services, profiles, processes, and groundings. Each pane lists all instances of the corresponding type. For the process instance pane we also use small icons next to the process names to distinguish the different types of processes (e.g., "a" for atomic and "c" for composite). An ontology containing multiple service descriptions would have several instances

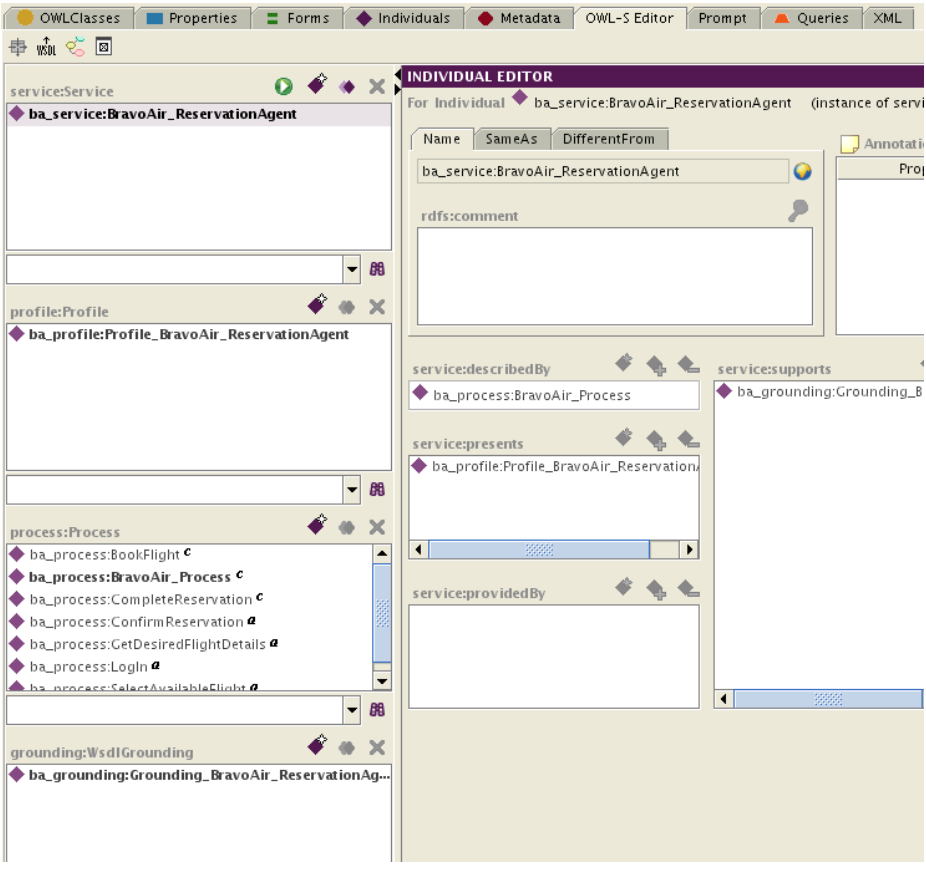

Fig. 1. The OWL-S Editor, a tab-widget plugin for Protégé is shown here next to the standard Protégé-OWL tabs to the left, and, to the right, other tab-widget plugins for ontology management, queries, and XML management

in each pane. To provide the user with an overview of how different service components fit together and which instances are related to one another (via `presents`, `describedBy`, and so on), the user can select an instance in one of the instance panes, and all instances that are directly related to the selected instance are emphasized in boldface in the other panes. In Figure 1 the service `ba_service:BravoAir_ReservationAgent` was highlighted in the service instance pane. As a result, its profile `ba_profile:Profile_BravoAir_ReservationAgent`, its top-level process `ba_process:BravoAir_Process`, and its grounding `ba_grounding:Grounding_BravoAir_ReservationAgent` are boldfaced in the other instance panes.

We have also implemented a graphical overview functionality. By selecting an instance and clicking a button in the toolbar, the user gets a graph view of the same information (see Figure 3).

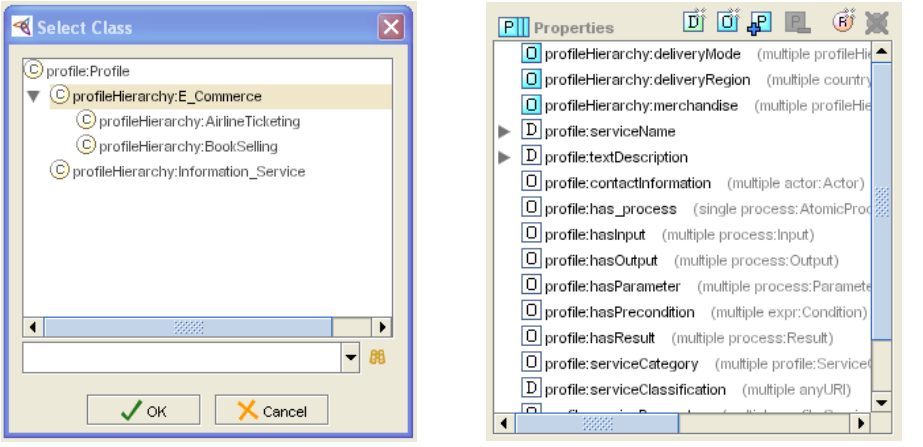

Fig. 2. A profile hierarchy for e-commerce that defines additional properties such as delivery mode and merchandise

When the user clicks an instance in one of the instance panes, the space to the right of the four panes changes to show a detailed *editing pane* for the selected instance. For example, if the user selects a profile instance, then the right window will show all properties of the profile. For some instances, such as processes, we have designed a layout on the right that provides a pictorial visualization of subprocesses, control constructs, and data flow (see Sections 3.3 and 3.4).

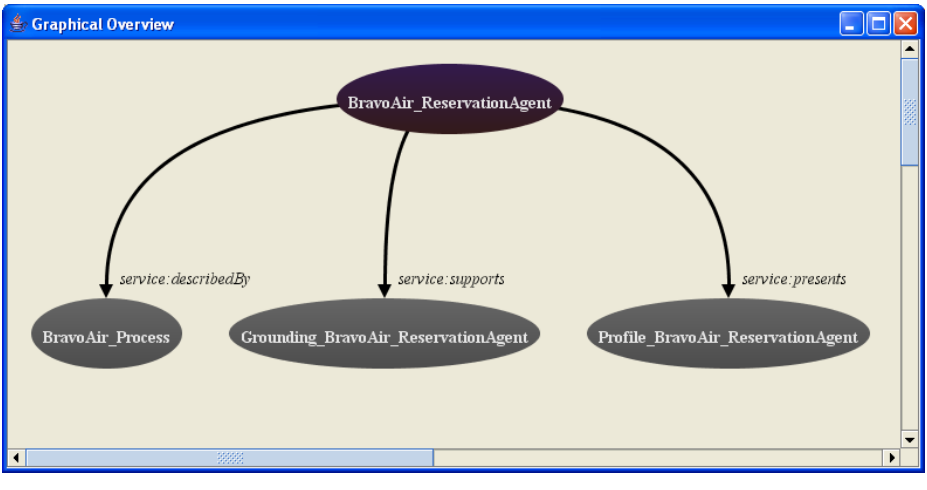

Fig. 3. Graphical Overview of the Bravo Air service

3.2 Managing Parameters

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`. As mentioned above, 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).

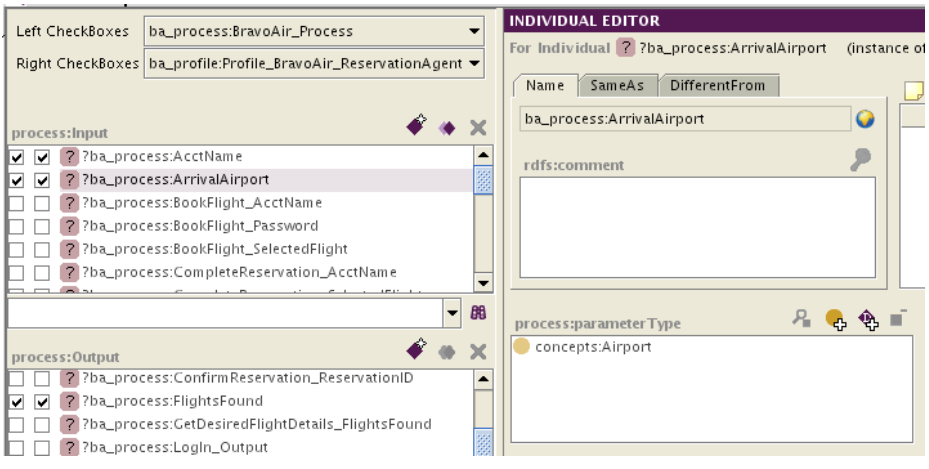

Fig. 4. IOPR Manager

To support efficient management of these IOPRs, we designed the *IOPR Manager*, which visualizes IOPR relationships in a very compact way (see Figure 4). Clicking a 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 (not shown here). The instance panes of the IOPR Manager show all the IOPRs in the KB, and allow the user to create and delete IOPRs. The user can also edit IOPR properties in the editing pane. In addition, two combo boxes at the top of the window allow users to select two processes and/or profiles that are to be compared with regard to their IOPRs. Associated with each combo box is a column of checkboxes, one for each IOPR. The user can simply check or uncheck these boxes to add or remove instances of the corresponding properties (`hasInput`, `hasOutput`, and so on).

As an example, if we select `ba_process:BravoAir_Process` and its profile `ba_profile:Profile_BravoAir_ReservationAgent` in the combo boxes, then the

checkboxes show that they both have the input `ba_process:ArrivalAirport`. In addition, the editing pane shows us that that the `parameterType` of this input is the `Airport` class (which is imported from a domain ontology of flight-related concepts).

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 3.5).

3.3 Control Flow

A powerful feature of OWL-S is the ability to model composite processes. 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 difficult to generate by hand or in a plain ontology editor not designed for this task. The OWL-S editor visualizes these control flows graphically, in a style similar to UML Activity Diagrams, 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 "work flow" graphs was a high priority for us. OWL-S control flows have more structure than arbitrary flow charts or UML activity diagrams, however. Therefore, we do not allow users to directly "draw" the work flow. Instead, we take advantage of the fact that all OWL-S control flows are trees in the graph-theoretical sense. We let the user model the control flow in a GUI tree component, with full drag-and-drop support, whereas the corresponding work flow graph is updated to reflect any changes to this tree (see Figure 5).

The view in Figure 5 is the editing pane for composite processes. If a user selects a composite process in the process instance pane, the editing pane to the right has the shown layout. If the user does a right-click on the process graph view of a composite process, a menu will pop up, offering zooming, printing, and SVG exporting capabilities.

As a further note, the process modeling that forms a part of the service semantics has reaped interest outside of the area of SWSs (e.g. in [6]). The process modeling part of our tool can be used to create process descriptions not necessarily related to WSs. However, this is not the primary goal of the OWL-S Editor.

3.4 Data Flow

In addition to control flow, composite processes can specify their data flow. For example, we can state that a certain input of Process B should be taken from a certain output of Process A. The goal in OWL-S (and our tool) is to also be able

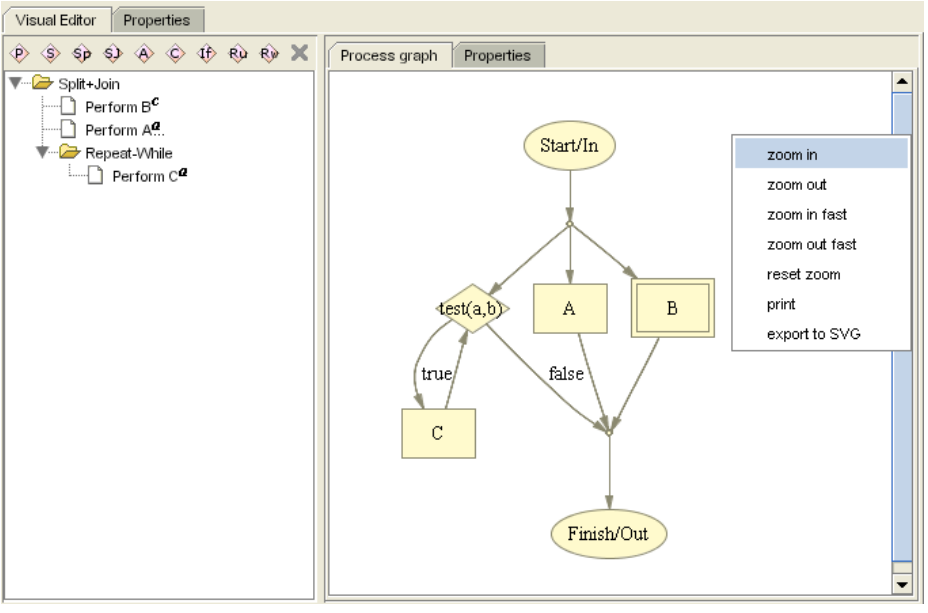

Fig. 5. A composite process, its tree structure shown to the left, and its graph representation to the right

to define more complex things, such as the input of Process B being the sum of the outputs of Processes A and C. The details of this remain to be worked out, but the simple one-to-one mappings should be sufficient for many applications.

Data flow is another area that is complicated to do by hand, but that can take great advantage of a graphical representation and specialized editing support. Both are supplied by the OWL-S Editor (see Figure 6).

Data flow definitions relate two parameters of different processes with each other. Either one associates a parameter of the parent process with a parameter of one of its component processes, or one relates two parameters of two component processes (atomic or complex) in the same parent process. If the user clicks on one of the Perform boxes in the process graph, a popup window (shown in the lower part of Figure 6) appears. This popup window shows the properties of that Perform, including any incoming data flow. Here, the user selects an input of the process (left part in the figure), and a source from which to take the value (right part). For the source, the user needs to first select the process ("From Perform") and then a parameter in that process ("From Parameter") to create a data flow declaration.

3.5 Grounding and WSDL Import

We have already mentioned that OWL-S descriptions can relate to WSDL files through groundings. OWL-S processes can relate to WSDL files in several ways

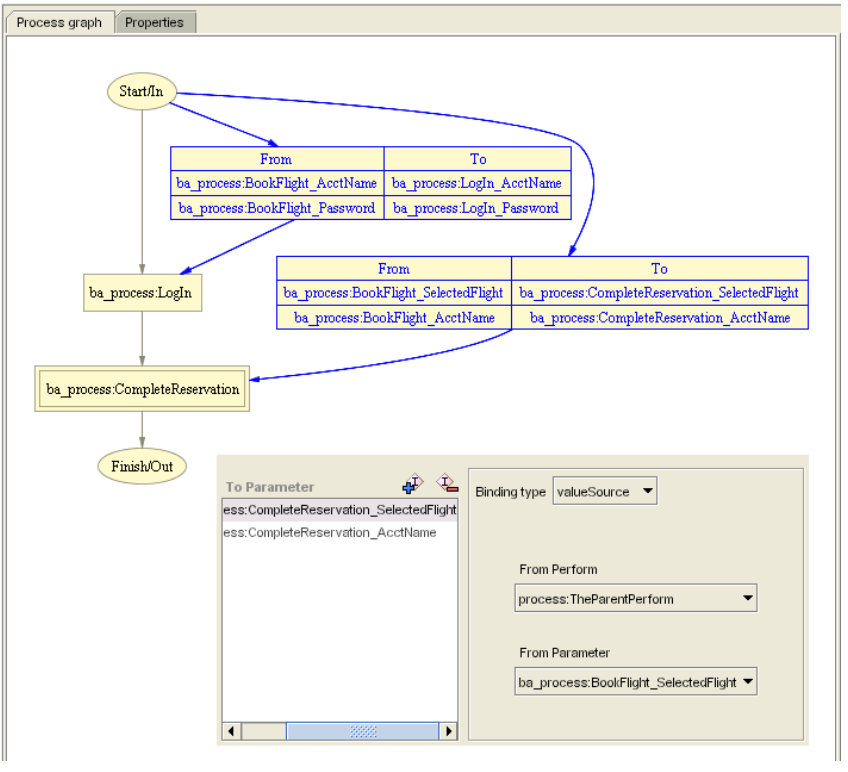

Fig. 6. Example of data flow between processes, and the popup window for editing data flow declarations

(see Figure 7), making it somewhat complicated to model this. 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 as well as 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 *WsdlMessageMaps*. In either of the two one-to-one correspondences, the WSDL service accepts serialized OWL, or the ontology operates on XSD[7] data types. 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. We have added rudimentary support for this task in the OWL-S Editor , but complex mappings still have to be written manually. It is our goal to make it straightforward and easy to declare these mappings in the OWL-S Editor.

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. To this end, we have integrated the WSDL2OWLS code, part of the OWL-S API

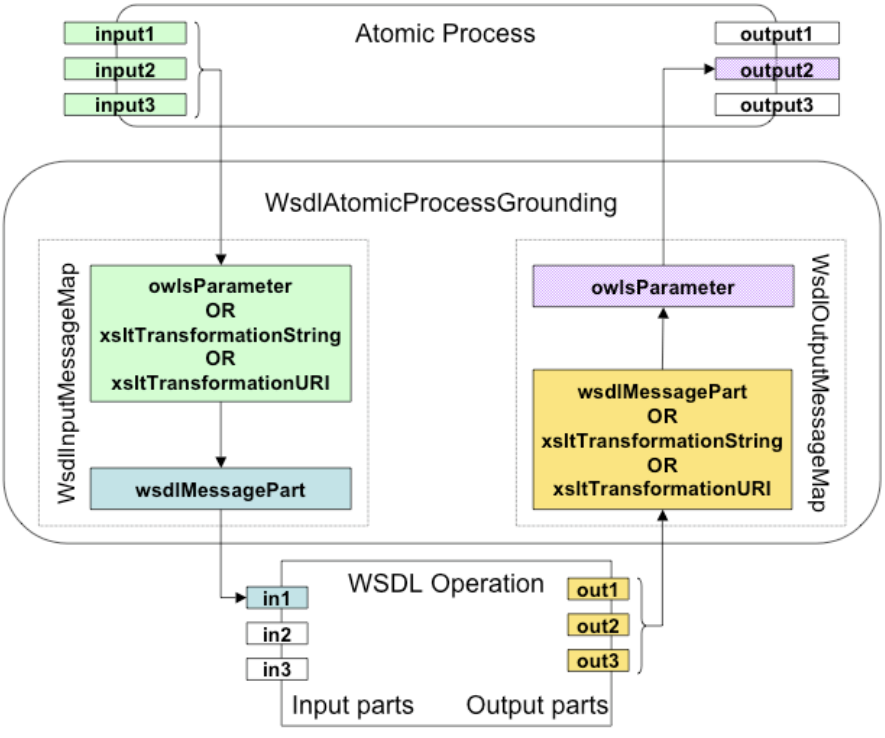

Fig. 7. Grounding: WSDL Message Maps

from Mindswap[8], to the OWL-S Editor. This allows users to perform this type of OWL-S generation from a WSDL file by clicking one of the toolbar buttons (see Figure 1).

3.6 Execution

An exciting feature of the OWL-S Editor is the ability to actually *execute* services inside the editing environment. Selecting a Service instance and clicking the 'play' button (see Figure 1) 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 (see Figure 8) 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. This functionality is work in progress, but 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.

[8] `http://www.mindswap.org/2004/owl-s/api/`

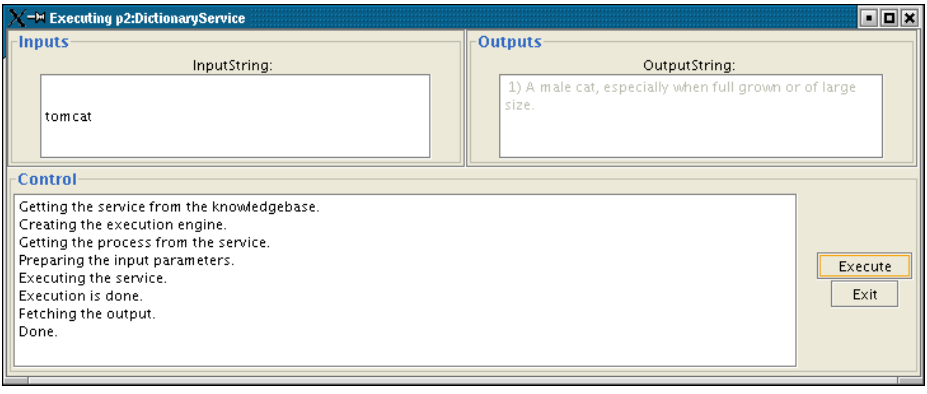

Fig. 8. Execution of a Semantic Web Service

4 Related Work

The OWL-S IDE project[9] is also concerned with the development of OWL-S services. The OWL-S IDE is a plugin for Eclipse[10], which attempts to integrate the semantic markup with the programming environment. Developers can write their Java code in Eclipse, and run a Java2OWLS tool to generate an OWL-S "skeleton" directly from the Java sources.

The idea of integrating SWSs more closely with the programming environment used to develop the service implementations is a good one. However, Eclipse does not support ontology editing, and there is no KB from which to choose the domain concepts to which the OWL-S files should relate. Furthermore, it will often be more useful to generate the semantic markup *before* the Java (or other) code, as the semantic descriptions can be seen as a higher level of abstraction of the programming modules. The OWL-S IDE does not provide any graphical visualization of services or processes.

There are plans to integrate Protege with Eclipse in the future, so perhaps we will have the best of both worlds—tight integration with the programming environment, as well as ontology editing and KB integration, all in the same IDE.

Another OWL-S Editor [8] has been developed at the University of Malta. It is a stand-alone program, providing WSDL import as well as a graphical editor and visualization for control flow and data flow. Not being integrated with an ontology editor, it shares some of the drawbacks of the OWL-S IDE, without gaining the advantage of programming-language integration.

ODE SWS is a tool for editing SWSs "at the knowledge level"[9], describing services following a Problem-Solving Methods (PSMs)[10] approach. OWL-S

[9] `http://projects.semwebcentral.org/projects/owl-s-ide/`, formerly known as CODE

[10] `http://www.eclipse.org`

plays a subordinate role in this environment, whereas the OWL-S descriptions are the main focus of our work.

IRS-3 [11] also follows the PSM approach, but lacks the graphical tools of ODE SWS or the OWL-S Editor, and does not support OWL-S, favoring WSMO instead.

To the best of our knowledge, none of the projects above have released their source code, whereas the source code for the OWL-S Editor has been available from the beginning, at `http://owlseditor.semwebcentral.org`.

5 Future Work

This tool represents early work in SWS design and development. In the following, we present some areas that we plan to work on in the future.

A limitation in Protégé is that it is not designed for concurrently working with multiple ontologies. However, it is often useful to be able to do so when working with SWSs. One often wants to edit service components spread across different subontologies, and the domain ontologies are normally separated from the service descriptions. Fortunately, the Protégé developers at working to implementing this functionality.

One aspect of OWL-S services not covered in this paper is the editing of preconditions and effects of processes, and conditions associated with control constructs such as If-Then-Else. In OWL-S, these are normally described in the SWRL language[11]. Currently, we simply provide a text box where users can enter these SWRL expressions. However, we plan to provide more user-friendly editing capabilities. Protégé has recently been enhanced with native support for SWRL, including a SWRL expression-builder, which will serve as the basis of this work.

A feature not yet implemented is online search for services. A central idea in OWL-S is reusability. The separation of service descriptions into process models, profiles, and groundings, means that these components can be re-used in other services. An online search capability for service components inside the OWL-S Editor would greatly facilitate such reuse. Such a search facility could also be used to find entire services, to be included as parts of a composite process that the user is working on in the OWL-S Editor. Ideally, the user should be able to give detailed search criteria, and find a service that matches her current needs (e.g. to find a service with inputs matching the outputs of previous processes in a composite process model). Various approaches to online searching could be implemented, ranging from brute-force Google[12] search for `.owl` files, via semantic search engines such as Swoogle[13], to service-specific systems such as semantically enhanced UDDI registries[12].

[11] `http://www.w3.org/Submission/2004/SUBM-SWRL-20040521/`
[12] `www.google.com`
[13] `www.swoogle.org`

We are also working on improvements to the graph overview presented in Section 3.1 to (1) show the entire "forest" of OWL-S instances simultaneously, and (2) allow users to change the relationships between the service components, i.e. add or remove instances of the properties `presents`, `describedBy`, etc.

We are also interested in the generation of OWL-S descriptions from BPEL4WS files. We are closely following work in this area[14] for possible inclusions in our tool.

6 Concluding Remarks

We have argued in this paper that Semantic Web Services (SWSs) could enable radically improved integration of businesses and networked communities, by automating service discovery, composition, and execution. SWSs thus promise great potential gains, but uptake has so far been slow. Lack of tool support has been a limiting factor for adoption of SWS technology.

OWL-S represents an emerging standard for SWSs, providing the concepts necessary to create detailed service descriptions. This paper has introduced the OWL-S Editor, a development tool for OWL-S services. This tool allows engineering of all aspects of SWSs, providing specialized views and design features wherever deemed necessary. The tool is well integrated with the Protégé OWL ontology editing framework. This integration means that developers can load, edit, and create domain ontologies, and subsequently relate their services to domain concepts in an easy and intuitive way.

Among the main features are graphical editing and visualization of control flow and data flow; the ability to easily maintain a large number of services; functionality to manage the relationships between service components and parameters; and generation of "skeletal" OWL-S descriptions from WSDL files.

A number of desirable extensions to the OWL-S Editor, such as online searching capabilities, and an integrated execution environment for services, have been discussed. In addition, we also plan to investigate the SWS software development process as a whole. This could involve such things as best practices for developing SWSs, "design patterns"[13] for SWSs, and the relationships between OWL-S and other representations and methodologies such as UML[14], Model-Driven Architectures[15], and PSL[16].

In providing this tool to the community, our aim is to make it easier to understand the concepts of SWSs, and to create semantic descriptions of services. We believe that this can bring a fruitful cross-pollination between practice and theory. As more people start developing SWSs, important feedback on using the service ontologies in various projects, and on design and implementation aspects of SWSs, could benefit the knowledge in this field.

The OWL-S Editor is available for download in both binary and source formats on `http://owlseditor.semwebcentral.org`. We welcome all feedback on our mailings list.

[14] `http://www.it.swin.edu.au/centres/cicec/bpel2owls.htm`

References

1. Erl, T.: Service-Oriented Architecture: A Field Guide to Integrating XML and Web Services. Prentice Hall, Upper Saddle River, NJ, USA (2004)
2. Berners-Lee, T., Hendler, J., Lassila, O.: The Semantic Web. Scientific American (2001)
3. McIlraith, S., Song, T., Zeng, H.: Semantic Web services. IEEE Intelligent Systems, Special Issue on the Semantic Web **16** (2001) 46–53
4. Martin, D., Paolucci, M., McIlraith, S., Burstein, M., McDermott, D., McGuinness, D., Parsia, B., Payne, T., Sabou, M., Solanki, M., Srinivasan, N., Sycara, K.: Bringing semantics to Web Services: The OWL-S approach. In: Proc. First Intern. Workshop on Semantic Web Services and Web Process Composition (SWSWPC 2004), July 6-9, 2004, San Diego, California, USA. (2004) `http://www.daml.org/services/owl-s`.
5. Knublauch, H., Fergerson, R., Noy, N., Musen, M.: The Protégé OWL plugin: An open developoment environment for semantic web applications. In McIlraith, S., Plexousakis, D., van Harmelen, F., eds.: Proc. 3rd Intern. Semantic Web Conference (ISWC 2004), Hiroshima, Japan, November 2004, Springer (2004) 229–243 LNCS 3298.
6. Schlenoff, C., Barbera, T., Washington, R.: Experiences in developing an intelligent ground vehicle (IGV) ontology in protégé. In: Proceedings of the 7th International Protégé Conference. (2004) `http://protege.stanford.edu/conference/2004/abstracts/Schlenoff.pdf`.
7. Fallside, D.C., (eds.), P.W.: XML Schema part 0: Primer second edition (2004) `http://www.w3.org/TR/xmlschema-0/`.
8. Scicluna, J., Abela, C., Montebello, M.: Visual modelling of OWL-S services. In: Proceedings of the IADIS International Conference WWW/Internet, Madrid, Spain, October 2004. (2004) `http://www.daml.org/services/owl-s/pub-archive/Visual-Modeling-of-OWL-S\%-Services.pdf`.
9. Goméz-Pérez, A., González-Cabero, R., Lama, M.: Development of semantic web services at the knowledge level. In: European Cnference on Web Services (ECOWS), Erfurt, Germany. (2004)
10. Fensel, D.: Problem Solving Methods. Springer-Verlag Telos (2000)
11. Domingue, J., Cabral, L., Hakimpour, F., Sell, D., Motta, E.: IRS-III: A platform and infrastructure for creating wsmo-based semantic web services. In: Proceedings of the Workshop on WSMO Implementations (WIW 2004) Frankfurt, Germany, September 29-30, 2004. (2004) CEUR Workshop Proceedings, ISSN 1613-0073.
12. Paolucci, M., Kawamura, T., Payne, T.R., Sycara, K.: Semantic matching of web services capabilities. In: Proceedings of the 1st International Semantic Web Conference (ISWC2002). (2002)
13. Gamma, E.: Design Patterns. Addison-Wesley, Boston, MA, USA (1995)
14. Brooch, G.: Object-Oriented Analysis with Design and Applications (2nd ed). Addison-Wesley, Boston, MA, USA (1993)
15. Raistrick, C., Francis, P., Wright, J.: Model Driven Architecture with Executable UML. Cambridge University Press. Cambridge, UK (2004)
16. Schlenoff, C., Gruninger, M., Tissot, F., Valois, J., Lubell, J., Lee, J.: The Process Specification Language (PSL): Overview and version 1.0 specification. NISTIR 6459, National Institute of Standards and Technology, Gaithersburg, MD. (2000)