OWL-S Editor [⋆]

Grit Denker[1] and Daniel Elenius[2] and David Martin[1]

[1] SRI International, California, USA
{grit.denker | david.martin}@sri.com
[2] Linköping University, Linköping, Sweden
daele@ida.liu.se

1 INTRODUCTION

An increasing number of organizations are endorsing Web Services (WS) technology to extend corporate resources to customers and partners and to leverage resources of others. Although Web Services, based on XML technology, allow interoperability, they do not support efficient and flexible search, allocation, composition, runtime monitoring, or invocation of those Web Services. *Semantic Web Services (SWSs)* use semantically rich annotations to facilitate these tasks. Richer semantics can provide fuller, more flexible automation of service provision, and use and support the construction of more powerful tools and methodologies.

OWL-S[3] [1], WSMO[4] and other related work may be viewed as efforts to lay the foundations for the most effective evolution of Web-Service-related capabilities that can be supported with current and maturing technologies. 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 need to facilitate service definition and annotation, execution and monitoring engines, service registration and discovery and much more. Our work is aimed at providing a flexible, yet powerful editor for OWL-S annotations. Our approach makes use of existing Protege knowledge representation technology and its OWL plugin while tailoring the OWL-S editor to the specific needs of service annotation. There exist other, similar OWL-S editor efforts, that—contrary to our effort—have not made any code available for download (e.g., the OWL-S editor developed at University of Malta[5] and CMU's Eclipse-based OWL-S editor[6]). Other related SWS efforts have different scope, such as the ODE SWS toolset[7] that uses Problem-Solving Methods instead of process-based service models.

In the remainder of this paper we will briefly introduce OWL-S in Section 2. We discuss the OWL-S editor architecture and describe the envisioned functionality of the editor in Section 3. We will also illustrate the current state of design and implementation. We conclude with future capabilities of the OWL-S editor in Section 4.

2 OWL-S

At the highest level, the goal of SWS technology is to maximize opportunities for effective automation of (all aspects of) Web-based service provision and use, thus reducing the manual configuration and programming effort that is necessary for online service use today.

The SWS approach to facilitate such automation is to use machine reasoning supported by ontologies. Domain ontologies from the Semantic Web describe the domain of the service (such as flights, tourism, e-business, books, etc), and an ontology of service concepts specifies in great detail how the service works and how it relates to its domain. OWL-S is such an ontology of service concepts. OWL-S organizes a service description into four conceptual areas: the *service*, the *profile*, the *process model*, and the *grounding*.

The service simply binds the other parts together into a unit that can be published and invoked. The profile provides a general description of the Web Service, intended to be published and shared to facilitate service discovery. Profiles can include both functional properties (inputs, outputs, preconditions, and results)

[⋆] Supported by the Defense Advanced Research Projects Agency through the Air Force Research Laboratory under Contract F30602-00-C-0168 to SRI.
[3] http://www.daml.org/services
[4] www.wsmo.org
[5] http://staff.um.edu.mt/cabe2/supervising/undergraduate/owlseditFYP/OwlSEdit.html
[6] http://projects.semwebcentral.org/projects/owl-sed/
[7] kw.dia.fi.upm.es/odesws/

and non-functional properties (service name, text description, contact information, service category and additional service parameters). The functional properties are derived from the process model. The process model describes how the service performs its tasks and it includes information about the service's inputs, outputs (including a specification of the conditions under which various outputs will occur), preconditions (circumstances that must hold before the service can be used) and results (changes brought about by the service). For a complex, composed service, the process model shows how it breaks down into simpler component processes, and the flow of control and data between them. The grounding specifies how the service is invoked, by detailing how the atomic processes in the process model map onto WSDL representations.

3 ARCHITECTURE, DESIGN, AND IMPLEMENTATION STRATEGY

The OWL-S editor is designed to be integrated as closely as possible with the Protege OWL plugin produced by Stanford Medical Informatics[8]. The editor is fully developed as open source and is hosted at the SemWebCentral site[9].

We envision an integrated set of capabilities supporting the editing of all four of the ontology areas (process model, grounding, profile, and the top-level service ontology). Knowledge about the relationships of the four areas can be used in several ways to assist the user: for example, by providing various kinds of consistency checking between the four areas, and by generating (full or partial) declarations in one area based on declarations already constructed in another area.

The general architecting strategy is that the OWL-S editor will accommodate a variety of "extenders[10]". For example, it should be possible to add on a variety of different matchmakers, so that, when constructing a composite process, a matchmaker can be consulted for a list of published atomic processes accepting or producing given inputs, outputs, preconditions, and results. To support this, we will provide a documented API and software mechanisms for plugging in an arbitrary matchmaker at runtime.

The main interface of the OWL-S Editor in Protege is a Protégé*tab widget*, which accommodates service-specific design capabilities as described in the following section.

3.1 OWL-S Tab for Protege

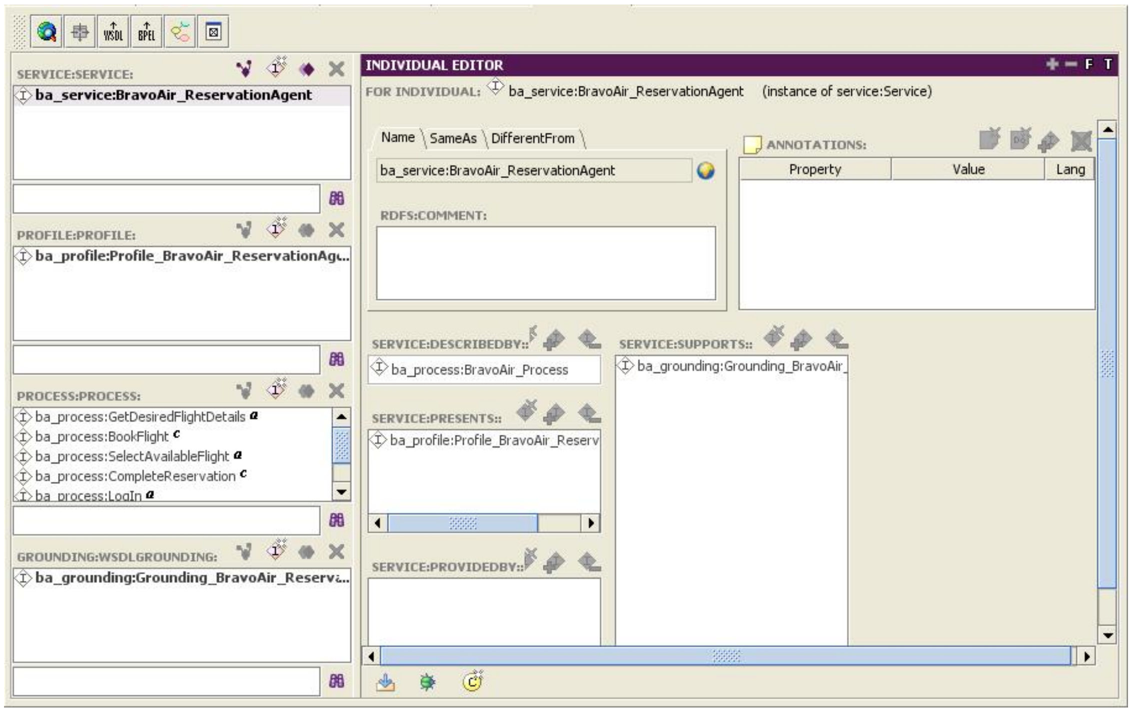
The OWL-S tab in Protege (see Figure 1) supports editing of all OWL-S subontologies, i.e., processes, groundings, profiles, and services. For each of the subontologies we created a pane (vertically arranged) that lists all instances of the corresponding type. The space to the right of the four panes shows detailed information about the subontology. For example, if the user selects a profile instance, then the right window will show all properties directly associated with profiles. There are thus four *editing modes*. Currently we are using existing Protege capabilities to display the properties of a class. In the future we will redesign the layout on the right. In particular, for composite processes we envision a graphical "workflow editor" that adequately visualizes the subprocesses and the control constructs used to combine them in a composite process. One of the main design principles of the OWL-S tab is to graphically support as well as possible the relationships among the various subontologies of OWL-S. If the user clicks on a service, profile, process, or grounding, the chosen instance is highlighted and all other instances that are related to the chosen instance via a direct relationship such as "presents" (between service and profile) or "describedBy" (between service and process) are automatically emphasized in boldface. This way the user can easily get an overview which service elements belong together. Moreover, the mode window for processes uses mini-icons to distinguish the different types of processes (e.g., "a" for atomic and "c" for composite).

The four icons on the top of the four panes will be used to implement an interface to (online) search functions (e.g., for the purpose of matchmaking), to bring up the IOPR manager, to generate OWL-S from existing WSDL and for other OWL-S menu options. Incorporating OWL-S-specific menus into the OWL-S tab has the advantage of leaving the Protege GUI unchanged for those users that do not intend to work with OWL-S.

[8] `http://protege.stanford.edu/plugins/owl`

[9] `http://owlseditor.projects.semwebcentral.org/`

[10] An extender can either be implemented as a Protege plugin, or in a more ad-hoc fashion defined by us in terms of Java interface mechanisms.



SERVICE:SERVICE: ba_service:BravoAir_ReservationAgent

PROFILE:PROFILE: ba_profile:Profile_BravoAir_ReservationAg...

PROCESS:PROCESS:
ba_process:GetDesiredFlightDetails *a*
ba_process:BookFlight *c*
ba_process:SelectAvailableFlight *a*
ba_process:CompleteReservation *c*
ba_process:LogIn *a*

GROUNDING:WSDLGROUNDING: ba_grounding:Grounding_BravoAir_Reserv...

FOR INDIVIDUAL: ba_service:BravoAir_ReservationAgent (instance of service:Service)

Name: ba_service:BravoAir_ReservationAgent

SERVICE:DESCRIBEDBY:: ba_process:BravoAir_Process

SERVICE:SUPPORTS:: ba_grounding:Grounding_BravoAir_

SERVICE:PRESENTS:: ba_profile:Profile_BravoAir_Reserv

Fig. 1. OWL-S in Protege

3.2 IOPR Manager

As mentioned above, inputs, outputs, preconditions, and results (IOPRs) are central to the characterization of services. For the purpose of service specification it is important that there are certain restrictions on the use of IOPRs in the four subontologies. For example, the IOPRs in the profile are a subset of the IOPRs in a process that is associated to the same service as the profile. During specification the user could choose to use the information of one of the subontologies to create (partial) information for another subontology, or the user might want to check consistency between all modes with respect to all defined restrictions. To succinctly support consistency and completeness checks for services, we designed and started implementing a special-purpose IOPR manager that visualizes in a very compact way IOPR relationships between all subontologies. Figure 2 depicts the overall design of the manager. The IOPR manager shows an overview of all IOPRs for selected process, profile, and WSDL grounding. The user can choose an atomic or composite process from the scroll-down menu. If the box next to a parameter is ticked off, then this parameter is defined in the subontology. The user can tick off parameters and press the update button in order to create new instances of the "hasParameter" properties in the chosen subontologies. Similarly, unticking boxes deletes the relationships, but does not delete the parameter declarations themselves. Creation and deletion of parameter is done using the icons "I" and "X" to the left. Clicking on a parameter brings up a separate window that allows editing the properties of a parameter. If the user chooses a composite process, then the WSDL Grounding row is greyed out. If the user chooses to update parameters, the IOPR manager will give warnings such as "Parameters in profile which are not in process" etc and give the user the choice to update anyway or modify her request.

4 FUTURE FEATURES

OWL-S includes three types of processes: *atomic*, *simple*, and *composite*. *Composite* processes are constructed from subprocesses, which in turn can be either atomic, simple, or composite. Control constructs such as *Sequence* and *If-Then-Else* are used to specify the structure of a composite process. In addition to describing control flow, this structural specification also shows which of the various inputs of the composite process

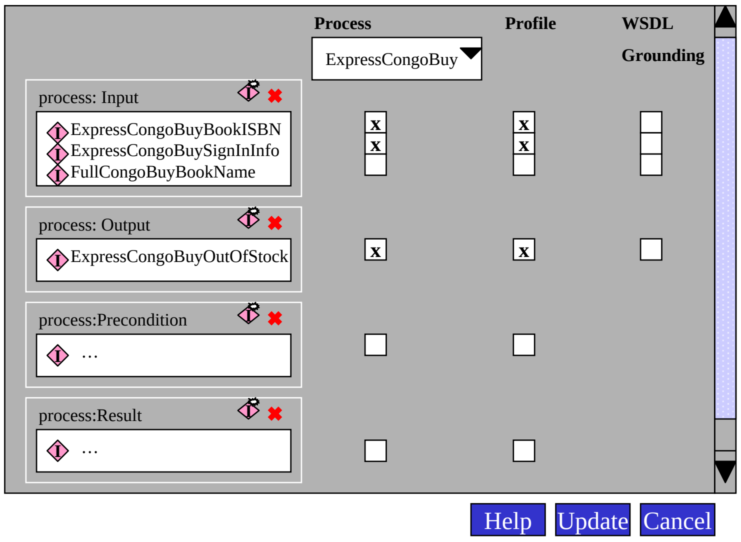


Fig. 2. IOPR Manager

are accepted by which of its subprocesses (and similarly for outputs). We have a first design to support a graphical interface for composite process specification, in particular illustrating the tree structure of complex processes as shown in Figure 3. As for extenders to the OWL-S editor, an OWL search/matchmaker plugin

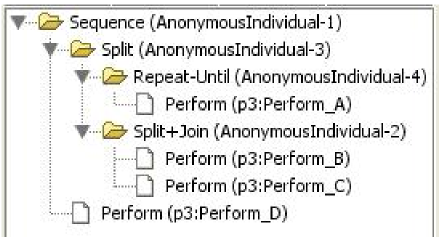

Sequence (AnonymousIndividual-1)
Split (AnonymousIndividual-3)
Repeat-Until (AnonymousIndividual-4)
Perform (p3:Perform_A)
Split+Join (AnonymousIndividual-2)
Perform (p3:Perform_B)
Perform (p3:Perform_C)
Perform (p3:Perform_D)

Fig. 3. Process Tree View

would be of considerable value. This would provide the means to search the Semantic Web for existing domain classes (typically for specifying types of inputs and outputs). The search plugin would display a list of relevant OWL elements, ordered by degree-of-match. The user would be able to select from these, and then the selected OWL-S elements would be loaded into a Protege knowledge base. It would also be possible to load the entire ontology to which a selected Ontology element belongs. Other examples where an extender strategy will be useful are DL reasoners, code generation modules, rules language editors, surface language parsers, and WSDL-generation components.

Another feature will extend the capabilities of generating OWL from Protege. The current OWL plugin allows to write ontologies to OWL code. For the OWL-S editor we will extend this with the capability of generating separate OWL files for a service, one for each subontology.

REFERENCES

1. 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)