

An Interactive Approach for Specifying OWL-S Groundings

Gerald C. Gannod¹²
Division of Computing Studies
Arizona State University – Polytechnic
Mesa, AZ 85212
gannod@asu.edu

Raynette J. Brodie, and John T.E. Timm
Dept. of Computer Science & Engineering
Arizona State University – Tempe
Tempe, AZ 85287-8809
{raynettejb, area51}@asu.edu

Abstract

OWL-S is an instance of the Web Ontology Language (OWL) that is used to describe and specify semantic web services. While OWL-S provides a promising mechanism for specification, publication, discovery, integration, and access, the learning curve can be high. Current practices in Web services tend to focus on lightweight specification using automated tools that generate WSDL descriptions. One of the advantages of OWL-S is its flexibility in allowing the creation of many groundings or bindings for a single semantic web service. In this paper, we propose an approach for generating groundings for a semantic web service and demonstrate how the use of lightweight interactive tools facilitates creation of groundings for a semantic web service.

1. Introduction

A *semantic web service* extends the capabilities of a Web service by associating semantic concepts to the Web service in order to enable better search, discovery, selection, composition and integration. The OWL-S specification language [1] has been created to provide a mechanism for describing domain concepts using ontologies while providing constructs for specifying information about how to compose and access services. While the benefits of semantic web services are attractive, they have yet to gain widespread adoption due to the lack of tools that support their specification and development. The current state of the practice for non-semantic web services is to develop the service using tools such as Visual Studio .NET and then automatically generate an appropriate WSDL specification. Imposing further requirements of adding OWL-S specifications in order to create semantic web

services offers many challenges including a learning curve that is often undesired by developers.

Model Driven Architecture (MDA) is an approach to software development that is centered on the creation of models rather than program code [2]. The primary goals of MDA are portability, interoperability, and reusability through an architectural separation of concerns between the specification and implementation of software. In MDA-based approaches, the focus is upon creation of software via the development of models specified using standard and wide-adopted languages such as the Unified Modeling Language (UML).

One of the advantages of OWL-S is its flexibility in allowing the creation of many groundings or bindings for a single semantic web service. We are developing an approach for constructing OWL-S specifications that focuses on two activities. First, we have developed an MDA-based approach for generating OWL-S *profile* and *process* specifications [1]. Second, we have developed an interactive approach for generating OWL-S *groundings*. This two part approach allows development of semantic web services to occur in two stages. In this approach, the semantic and architectural concerns associated with specifying semantic web services can be performed by software and knowledge architects. The mapping of Web services described using WSDL to operations contained in the profile and process specifications are intended to be performed by developers of web services as they are constructed or by architects as they identify existing services that meet the intended behaviors of the semantic services.

We provide an overview of the general framework being developed to support construction of OWL-S specifications and describe the technique being used to generate OWL-S groundings. Construction of the OWL-S profile and process specifications relies on the use MDA and transformations of XMI [3] (e.g., XML representations of UML) using XSLT. In this paper, we present a framework for OWL-S groundings and an interactive tool for generating groundings.

The remainder of this paper is organized as follows. Section 2 describes background information relevant to

¹ Contact Author.

² This author supported by National Science Foundation CAREER grant No. CCR-0133956.

semantic web services. Section 3 describes our general framework as well as the details of the interactive tool for constructing semantic web services. An illustrative example using our approach is presented in Section 4. Related work is discussed in Section 5 and finally, Section 6 draws conclusions and suggests future investigations.

2. Background

This section describes background material in the area of web services, ontologies and OWL-S.

2.1. Web Services

A web service is a modular, well-defined software component that exposes its interface over a network. Applications use web services by sending and receiving XML messages over HTTP as shown in Figure 1. Web services provide the foundation for loosely coupled, service-oriented software systems. They allow multiple organizations to interact in a uniform, well-defined manner. This is a major step towards interoperability between multiple heterogeneous distributed systems.

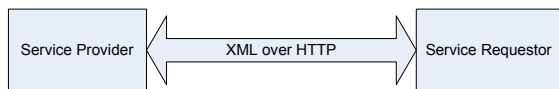


Figure 1 Web service interaction between requestor and provider

Web service interfaces are defined using the Web Service Description Language (WSDL) [4]. WSDL is an XML-based language for describing the interface of a web service including message types and bindings. WSDL does not provide any sort of formal semantic description of the web service. In order to provide semantics for web services, they must be enhanced using ontologies.

2.2. OWL-S

An ontology is a set of concepts, their properties, and the relationships between them. Ontologies provide the building blocks for expressing semantics in a well-defined manner [5]. A simple ontology example is shown in Figure 2. Ontologies serve as the primary building block for adding semantics to web services. Modeling with ontologies is not all unlike domain modeling in the software engineering context. Several analogies exist between software modeling and knowledge engineering including the use of classes, inheritance, properties, etc. Unfortunately, ontologies have not been widely accepted in general software development.

The Web Ontology Language (OWL) is an XML-based language for describing ontologies [5]. The OWL was designed to allow for the specification of semantic

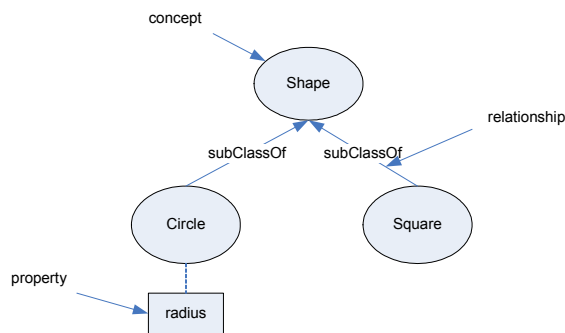


Figure 2 Simple Example of an ontology

descriptions for resources on the Web, which could be interpreted by autonomous software agents. These resources can be anything from simple web pages to web services. Because the syntax of OWL is XML, it is platform-independent, can be easily transferred over a network, and manipulated with existing automated tools. OWL-S is an ontology for services created by the DAML group [1]. The ontology is broken into three parts. The *Service Profile* describes the capabilities of the service. The *Service Model* describes how the service works internally. Finally, the *Service Grounding* describes how to access the service. The OWL-S ontology is useful in that it provides a uniform mechanism for describing the semantics of a web service.

3. Approach

This section describes the approach that we are developing to support the construction of OWL-S specifications including the interactive technique for creating OWL-S groundings.

3.1. Overview

Figure 3 shows the process workflow for our approach. The framework uses a model-driven development approach for specifying, mapping, and executing semantic Web services. In this regard, the framework not only allows the specification of semantic Web services but facilitates the mapping of constructs in those semantic descriptions to concrete service realizations. These concrete service realizations are transformed into executable specifications for infrastructures such as a BPEL [6] execution engine. In this framework, an architect is responsible for creating models using UML. The framework manages the transformation process to OWL-S with very little additional effort on behalf of the developer. This benefits the developer in two ways. First, the developer is able to focus on creating models instead of writing code or in the case of semantic Web services creating a semantic specification. Secondly, the developer

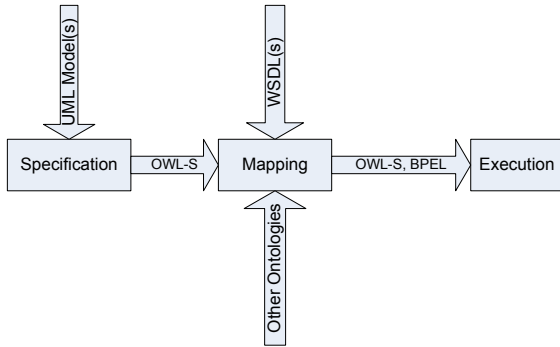


Figure 3 Specification Framework

becomes more efficient because low-level details are abstracted away and the developer can focus more on the top-level structure and semantics. The developer can leverage existing skills in popular UML tools such as Poseidon [7] and Eclipse [8].

The second stage of the framework involves a tool being developed that automates the process of mapping concepts in the OWL-S description to concepts in the WSDL file of a concrete service realization. This tool uses the process model portion of the OWL-S description as well as a set of WSDL files for corresponding services to facilitate generation of the OWL-S grounding portion of the OWL-S description.

Once the grounding is created, the final step in the model-driven development process involves execution of the OWL-S specification. In order to leverage existing technologies, a tool is being developed, as part of the general framework, to automatically generate an executable BPEL specification [6]. Using BPEL we can leverage existing execution engines. Semantic discovery and composition can take place utilizing the generated OWL-S specification. Once a service is composed, the BPEL specification can be used for execution. A semantic Web service composition may have multiple

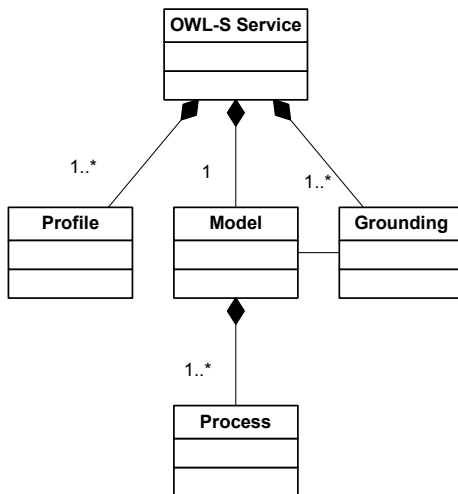


Figure 4 Structure of OWL-S Specification

concrete realizations from which completely separate executable specifications can be generated. Concrete realizations are typically traditional Web services described using WSDL but have the potential to be other kinds of accessible services.

3.2. Groundings

In this section we discuss issues related to construction of groundings. Figure 4 shows the conceptual structure of OWL-S specifications. Fully defined OWL-S service specifications are composed of a service description, a model, and many profiles and groundings. As described above, our framework covers specification of all of the entities shown in Figure 4. Figure 5 contains a diagram depicting the context of OWL-S groundings. At the foundation is OWL. OWL is used to define ontologies for the semantic web as well as the structure of OWL-S. OWL-S and accompanying ontologies provide the infrastructure by which semantic web services are defined. Finally, OWL-S groundings provide the mechanism by which OWL-S semantic web services are mapped to executable services. The stakeholders involved in activities related to each of the various levels of technologies are shown on the right. OWL and OWL-S are primarily under control of standards committees, while knowledge engineers are involved in the development of OWL ontologies. Service and application architects are expected to be the developers of OWL-S service descriptions, while groundings of those services are likely to be performed by application developers.

3.2.1. Mappings

There are two different perspectives that can be taken when creating OWL-S groundings. The first perspective comes from the viewpoint of software and knowledge architects that construct OWL-S specifications for semantic web services. This viewpoint can be considered to be a *topdown* view. From their perspective, the creation of an OWL-S specification is an activity of defining some abstract semantic service followed by the identification of services that realize and implement the semantic service. From this perspective, it is desirable to identify groundings for all of the abstract services described in the OWL-S specification. Also, it is possible that a wide variety of services may satisfy the behavior

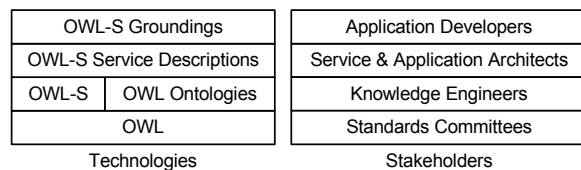


Figure 5 OWL-S Groundings Context

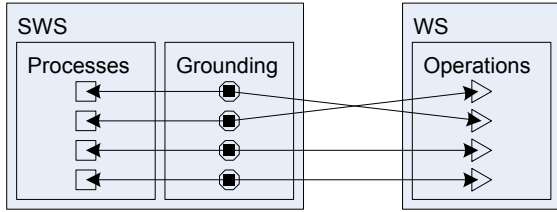


Figure 6 Complete Grounding to Single Service

required by the semantic service and thus many groundings could be specified. In this context, it may be expected that the mapping operation will create a complete grounding, so that all of the behavior required by the semantic service is identified and bound to the service. Accordingly, we offer the following definitions:

Definition: Complete Grounding. Let s be an OWL-S specification for some semantic service, and $s.Op$ be the set of all operations defined in $s.profile$ (a profile of s) and $s.model$ (the service model of s). A grounding $s.grounding$ for s is said to be *complete*, denoted $complete(s, s.grounding)$, if for all operations o in the set $s.Op$, there is a well-defined mapping to a concrete realization of o .

Definition: Well-defined mapping. Let s be an OWL-S specification for some semantic service and c be a Web service. A mapping from an operation o in $s.Op$ to an operation p in $c.Op$ is well-defined if there is a bijection mapping the inputs and outputs of o to the inputs and outputs of p .

A mapping is well-defined if all of the inputs and outputs of a given operation in the set $s.Op$ described above are mapped to a corresponding input and output of a concrete realization of the operation. We currently only handle cases where mappings are well-defined. However, there are many cases where interfaces may be mismatched, thus requiring *adaption* and *mediation*.

Figure 6 graphically represents one kind of complete grounding. In left side of the diagram shows a semantic Web service (labeled “SWS”) while the right side of the diagram shows a conventional web service (“WS”). Conceptually, among other things, both semantic web services and conventional web services specify a number of operations that are supported. In the diagram, the operations supported by the semantic web service are shown with a square while the conventional web service operations are shown with a triangle. The diagram itself shows that a grounding maps operations in SWS to operations in WS. In this case, the grounding maps all operations to the single service.

Figure 7 also shows a complete grounding but in this case, depicts a grounding that maps operations across several web services rather than just a single service. This grounding is complete since all operations are mapped to some operation in a conventional Web service.

A second perspective for constructing groundings comes from the viewpoint of a service provider that may only be providing part of the behavior expected in a semantic service. This viewpoint can be considered a *bottom up* view. For instance, consider the scenario of an e-commerce air travel ticketing service described as a semantic service in OWL-S. The service would include capabilities specific to the air travel domain such as searching for and reserving tickets for air travel. Other capabilities, however, may include general B2C (business to customer) operations such as shopping cart and credit card services. An OWL-S description of the e-commerce service could be specified with no specific groundings in mind so that organizations that do provide the fine-grained services could reuse the description, or rather “offer up” their services as part of a collaborative effort to provide a composite service. In this scenario, it would be conceivable that a service provider for a specific component of the semantic service would generate a grounding just for the portion that they provide. Since they may not have knowledge to complete the grounding for what their collaborators provide, the grounding specification would be incomplete. In this context, we define an *incomplete grounding* as follows.

Definition: Incomplete Grounding. Let s be an OWL-S specification for some semantic service, and $s.Op$ be the set of all operations defined in $s.profile$ (the profile of s) and $s.model$ (the service model of s). A grounding $s.grounding$ for s is said to be *incomplete*, denoted $\neg complete(s,$

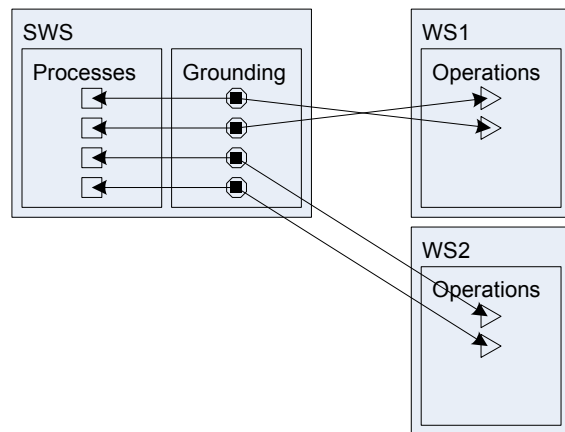


Figure 7 Complete Grounding to Multiple Web Services

$s.grounding$), if there exists an operation o in the set $s.Op$ for which there is no well-defined mapping to a concrete realization of o .

Figure 8 depicts an incomplete grounding. As shown in the diagram, an incomplete grounding is simply a grounding that lacks defined mappings for at least one operation in the semantic service.

The definition for incomplete groundings gives rise to a requirement that in order for an OWL-S specification to be “executable”, amongst the many incomplete groundings that may be created for a service there must be a complete grounding made up of mappings in the incomplete groundings. The notion of an executable OWL-S specification is given below.

Definition: Executable OWL-S Specification. Let s be an OWL-S specification for some semantic service and G with $|G| \geq 1$, be the set of all groundings for s . Specification s is *executable* if:

- a. $\exists g \in G : complete(g)$, or
- b. $\exists g' \notin G : \forall m \in g' (\exists g \in G : m \in g \wedge complete(s, g'))$

Clause a. states that if there is a complete grounding for a semantic service, then the OWL-S specification is executable. The second clause says that if there is a grounding that can be induced (e.g., constructed by taking members from the groundings of the service) that is itself complete, then the specification is executable.

Figure 9 shows a diagram depicting several groundings for a single semantic service. In this case, no single grounding is complete. However, in amongst the union of the groundings, a complete grounding does exist.

The fact that many groundings can be created and induced for a given OWL-S specification has some desirable benefits. Specifically the existence of multiple groundings introduces a certain degree of variability into a semantic service in such a way that each grounding potentially provides slightly different behavior. As such, special care must be taken to control how a given concrete service is selected when many groundings are present. In our work we are interested in utilizing product line concepts for selecting features

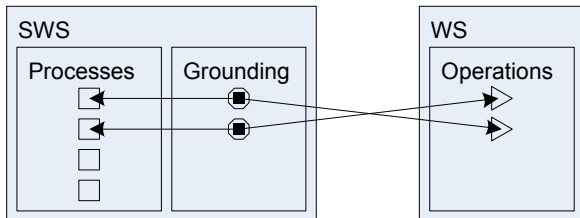


Figure 8 Incomplete Mapping

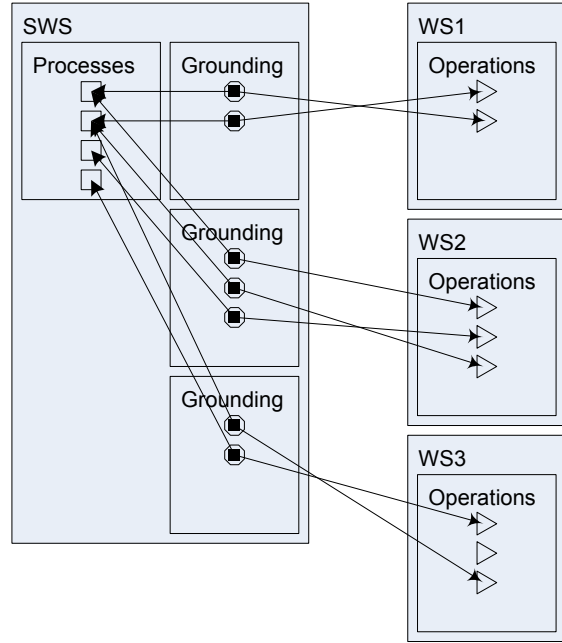


Figure 9 Executable Specification

with *feature diagrams* and then resolving conflicts using established techniques from the feature modeling domain.

The tool that we have developed to support the specification of groundings for semantic services supports both perspectives as described above by allowing a user to choose a view for describing mappings. In the case of the top down perspective, a user is presented with a list of OWL-S operations that must be mapped to services in WSDL specifications. In the case of the bottom up perspective, a user is presented with a list of WSDL operations that must be mapped to operations in an OWL-S specification.

3.2.2. Conceptual Architecture

The conceptual architecture for the grounding tool is shown in Figure 10. The operations are essentially focused on three primary operations: parsing, interactive mapping, and grounding synthesis. The inputs to the system vary based on the perspective taken by a user.

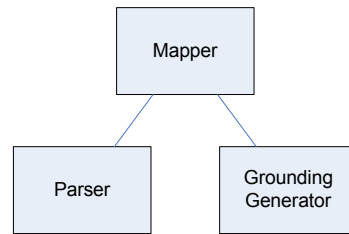


Figure 10 Conceptual Architecture

When taking the top down perspective, the inputs are an OWL-S specification and either a list of WSDL specifications or a list of search criteria for locating the WSDL specifications. The output for the tool in this perspective is typically a complete OWL-S grounding, although it can be incomplete if the mapping activity is either unfinished or the WSDL specifications do not cover all operations in the OWL-S specification.

When taking the bottom up perspective, the inputs are an OWL-S specification and a single WSDL specification. In this case, the OWL-S specification can be optional as long as some mechanism exists to locate an appropriate OWL-S specification through some search mechanism. The output for the tool in this perspective is again an OWL-S grounding, but in this case is likely to be incomplete except in those case that the single WSDL file completely covers all of the operations found in the OWL-S specifications provided as input or through search.

The mapping activity, controlled by the *mapper* component in the diagram, is interactive in the sense that a user must identify the appropriate mappings between abstract OWL-S operations and concrete WSDL operations. We have chosen to develop the tool to use an interactive approach rather than automatic one as an initial solution for generating groundings. We plan on developing a heuristic approach based on the use of search techniques that we have developed previously [9]. Currently the tool supports both input perspectives and allows a user to create mappings for atomic processes in an OWL-S specification and single WSDL specifications. The search capabilities are being built into the system in a future revision.

Once a grounding has been completed using the tool, it is combined with an appropriate OWL-S specification, depending on the input context. In order to verify that the grounding is correct both syntactically and semantically (with respect to an ontology) we use the Protégé system [10]. Protégé will verify syntactic correctness as well as check that semantic constraints specified in the OWL-S specification are not violated by the generated grounding and that the entire OWL-S specification meets general constraints of the referenced ontologies.

3.2.3. User Interface Design

Figure 11 shows a screenshot of part of the user interface for the mapping tool. This particular instance of the tool is shown from the bottom up perspective where a WSDL specification containing several operations is being mapped to an OWL-S specification.

The OWL-S Mapping provides a simple and intuitive interface for users to generate a grounding instance from a WSDL and an OWL-S specification. We introduction to the features of the interface described in the context of the example in Section 4.

The interface is divided into two symmetrical halves, the left pane and the right pane. In this example the user selects to map a WSDL to an OWL-S process. This perspective shows the WSDL in the left pane and the process file in the right pane.

Iterative steps to grounding generation:

1. Select a WSDL operation name and Process operation name.
2. View properties and attributes of each.
3. Align the inputs and outputs into a complete mapping.
4. Click “Apply Mapping” and repeat 1-4 as desired.

The selection area of the left pane shows each process in the WSDL, the inputs and outputs for that process, and the data types of the inputs and outputs. The selection area is designed as a tree view so the user can view the processes and associated attributes without restriction on navigation. Once the user finds a WSDL operation to include in this grounding instance, they click on the operation name to display the operation inputs, outputs and associated data types in the mapping area. Only one operation may be selected at a time. The currently selected operation name is shown above the left-pane mapping area.

The selection area of the left pane shows each concept in the OWL-S process file, the inputs and outputs for that concept, and the data types of the inputs and outputs. This area, like the left pane, supports free navigation. A user clicks on the concept name to display the inputs, outputs and associated data types in the mapping area. Only one concept can be selected at a time. The currently selected concept name is shown above the right-pane mapping area.

The mapping area is central on the interface of the tool. Here the user decides how the inputs and outputs of the selected WSDL operation correspond to the inputs and outputs of the selected Process operation. This mapping is realized when the user interactively arranges the input and output names so that the first row in the left pane maps to the first row in the right pane, and so on with each row.

A complete grounding requires that all inputs and outputs in the WSDL be mapped to by an input and output in the Process. To facilitate complete groundings, the left-pane mapping area is not modifiable by the user. Conversely, the right-pane mapping area provides the user with buttons to modify row location or delete rows.

Once the user is satisfied with the mapping of inputs and outputs from the selected WSDL process to the selected Process concept, the user clicks the “Apply Mapping” button.

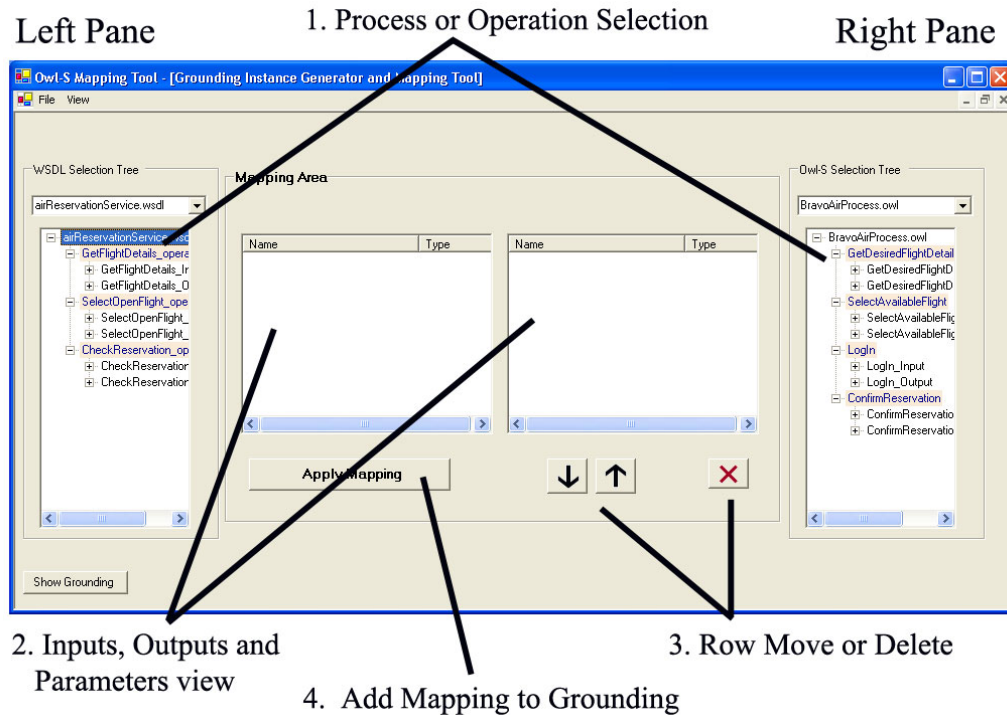


Figure 11 User Interface

A grounding contains a definition for each WSDL operation that is mapped to each OWL-S operation. Groundings will most often contain several definitions. The user will add a definition to their grounding by repeating the iterative steps to generation grounding.

The four steps in the grounding generation process are consistent with all mapping perspectives in the tool. These steps assume that the user chooses to map one WSDL file to one OWL-S process file. It is also necessary for the user to decide if the inputs and outputs are mapped in a way that makes semantic sense.

Currently the tool supports complete groundings to a single web service. Development of this tool is proceeding and planned improvements are underway. Future versions of the tool will support complete groundings to multiple Web services.

In order to validate this system, we are planning to perform a user study in order to compare this tool with others that have been created including the Meteor-S Web service composition tool [11].

3.2.4. Implementation

The tool was created using Visual Studio .Net 2003 and the .Net Framework. The parsing logic incorporates libraries from the System.Xml API; including XPathDocument, XPathNavigator, XPathExpression, and XPathIterator. The interface was constructed with the System.Windows.Forms API using C# as the code behind. Each pane on the interface contains a TreeView

for process selection and a ListView for a detailed process view. The grounding is displayed in a RichTextBox.

4. Example

In this section, we present an example demonstrating a usage scenario for the grounding tool. In this example, the user wants to generate a grounding with the WSDL file `airReservationService.wsdl` and the OWL-S process file `AirBravoProcess.owl`. Figure 12 shows a session corresponding to the BravoAir example with the output of the session appearing in Figure 13.

The WSDL operation `GetFlighDetails_operation` is identified in the WSDL treeview. The user clicks on the operation name in order to view the operation inputs, outputs and parameters in the mapping area. The OWL-S operation `GetDesiredFlightDetails` is identified and selected from the OWL-S treeview. The inputs, outputs and parameters of `GetDesiredFlightDetails` are shown in the mapping area.

In order to map the WSDL operation `GetFlightDetails_operation` with the OWL-S operation `GetDesiredFlightDetails`, the user needs to align the corresponding rows in the mapping area. In the left pane, the second row displays the WSDL parameter `departDate`. In the right pane, the second row displays the OWL-S parameter `DepartureAirport`. These parameters are not properly aligned. Certain rows in the right pane need to be moved so that the parameters in

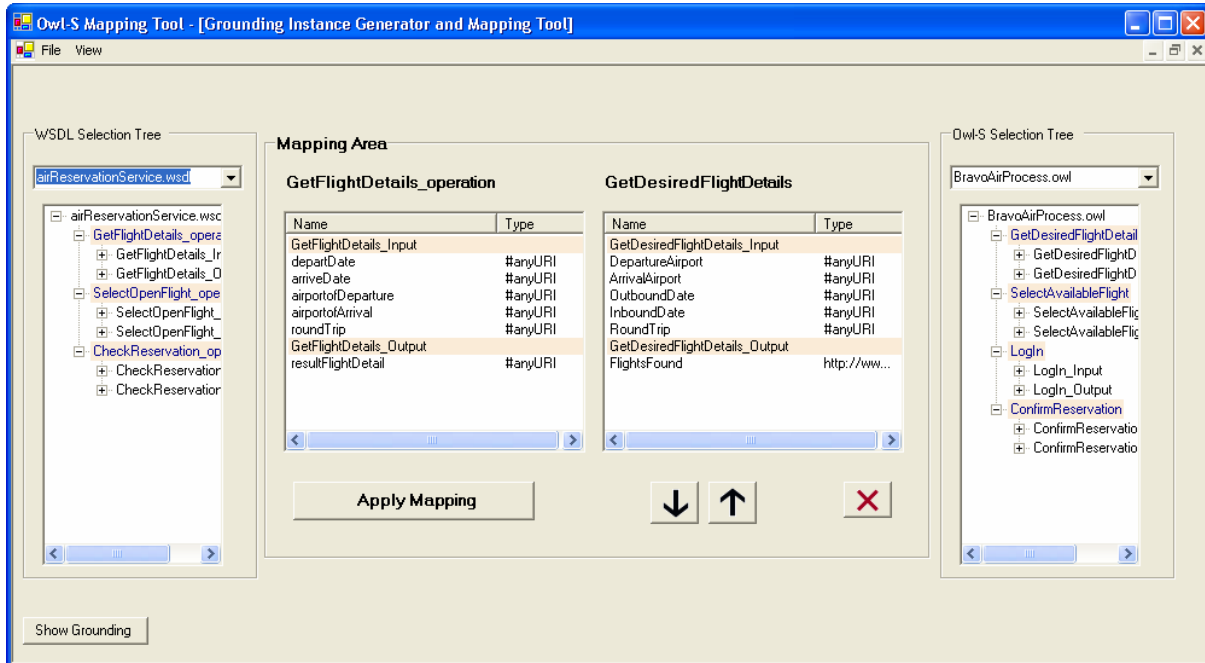


Figure 12 BravoAir Grounding Session

the left pane match in a way that makes semantic sense and creates a grounding that is correctly executable.

To map departDate, in the left pane, with OutboundDate, in the right pane, the user selects OutboundDate and clicks the up arrow twice. Now departDate and OutboundDate are both aligned in row two of the mapping area. The user moves other rows in the right pane as necessary to map each parameter. The user continues until arriveDate is in the same row as InboundDate, airportofDeparture is in the same row as DepartureAirport, and so on. In this operation mapping example, no rows need to be deleted.

Once the rows are mapped as desired, the user clicks the Apply Mapping button to add these details to the grounding in generation. If the user wants to view the instance just generated, they click on the Show Grounding button.

To add more operation mappings to this grounding, the user iterates through another cycle of identifying operations, viewing the operation details in the mapping area and row matching via arrow buttons. The Apply Mapping button will add operations to the grounding. The grounding view shows the operation just generated and a grounding instance with each mapped operations defined in the hasAtomicProcessGrounding tags.

Once the user is satisfied that the grounding contains each operation they want mapped from airReservationService.wsdl and AirBravoProcess.owl, they save the grounding with a specified name to a specified location. For the grounding to be executable,

the user must update the URIs for the original WSDL and OWL-S files in the grounding as needed.

5. Related Work

Currently, there are only a few tools which support mapping between semantic web service descriptions and concrete service realizations. These tools are outlined below.

The OWL-S Editor supports visual editing of OWL-S descriptions [12]. The tool provides a graphical user interface. In general, the tool generates an OWL-S description entirely from a WSDL specification. Therefore, no direct user input is required to do mapping. This approach differs from the one discussed in this paper as it maps from WSDL to OWL-S but does not support the reverse direction. The tool in this paper starts with an OWL-S specification that may be generated manually or using some automated tool as in [3]. The specification may or may not have a concrete grounding. Our tool interprets the OWL-S profile and process and allows for a manual mapping between concepts in the OWL-S process and those in the WSDL file. This is particularly useful when creating multiple grounding from a set of existing services.

The METEOR-S project is a comprehensive effort that focuses on the creation, management and execution of Semantic Web services [11]. Part of the effort focuses on the mapping of services by taking a set of service ontologies, written in DAML, and visually displaying them to the user. The user can then go through the process of either manually or semi-

automatically mapping of concepts to services. The semi-automatic mapping process uses structural and linguistic mapping algorithms to map concepts between ontologies and WSDL. This tool relies on algorithms to provide automated mapping, something which our tool currently does not support. However, their tool does not use OWL-S but rather relies on DAML as its description language. The user interface requires

intimate knowledge of both the ontology structure and the WSDL concepts and presents a significant amount of details to the user. Our tool focuses on simplifying the mapping process by abstracting away unnecessary details about both the OWL-S descriptions and the WSDL service specifications.

Jaeger, Engel and Geihs [13] have proposed a methodology for developing semantic descriptions of

```

<grounding:Wsd grounding:WsdGrounding rdf:ID="Grounding_BravoAir_ReservationAgent"/>
  <service:supportedBy rdf:resource="Grounding_BravoAir_ReservationAgent" />
  <grounding:hasAtomicProcessGrounding rdf:resource="#Wsd grounding:WsdGrounding_GetFlightDetails"/>
</grounding:Wsd grounding:WsdGrounding>

<grounding:Wsd grounding:WsdAtomicProcessGrounding rdf:ID="Wsd grounding:WsdGrounding_GetFlightDetails">
<grounding:wsdlDocument rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">
  C:\airReservationService.wsdl</grounding:wsdlDocument>
<grounding:owlsProcess rdf:resource="C:\BravoAirProcess.owl#GetFlightDetails"/>
<grounding:wsdlOperation rdf:resource="GetFlightDetails_operation"/>
<grounding:wsdlInputMessage rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">
  C:\airReservationService.wsdl#GetFlightDetails_Input
</grounding:wsdlInputMessage>

<grounding:wsdlInput>
  <grounding:Wsd grounding:WsdInputMessageMap>
    <grounding:owlsParameter rdf:resource="C:\BravoAirProcess.owl#OutboundDate"/>
    <grounding:wsdlMessagePart rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">
      C:\airReservationService.wsdl#departDate
    </grounding:wsdlMessagePart>
  </grounding:Wsd grounding:WsdInputMessageMap>
</grounding:wsdlInput>
<grounding:wsdlInput>
  <grounding:Wsd grounding:WsdInputMessageMap>
    <grounding:owlsParameter rdf:resource="C:\BravoAirProcess.owl#InboundDate"/>
    <grounding:wsdlMessagePart rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">
      C:\airReservationService.wsdl#arriveDate
    </grounding:wsdlMessagePart>
  </grounding:Wsd grounding:WsdInputMessageMap>
</grounding:wsdlInput>
<grounding:wsdlInput>
  <grounding:Wsd grounding:WsdInputMessageMap>
    <grounding:owlsParameter rdf:resource="C:\BravoAirProcess.owl#DepartureAirport"/>
    <grounding:wsdlMessagePart rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">
      C:\airReservationService.wsdl#airportofDeparture
    </grounding:wsdlMessagePart>
  </grounding:Wsd grounding:WsdInputMessageMap>
</grounding:wsdlInput>
<grounding:wsdlInput>
  <grounding:Wsd grounding:WsdInputMessageMap>
    <grounding:owlsParameter rdf:resource="C:\BravoAirProcess.owl#ArrivalAirport"/>
    <grounding:wsdlMessagePart rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">
      C:\airReservationService.wsdl#airportofArrival
    </grounding:wsdlMessagePart>
  </grounding:Wsd grounding:WsdInputMessageMap>
</grounding:wsdlInput>
<grounding:wsdlInput> ... </grounding:wsdlInput>
<grounding:wsdlOutput>
  <grounding:Wsd grounding:WsdOutputMessageMap>
    <grounding:owlsParameter rdf:resource="C:\BravoAirProcess.owl#FlightsFound"/>
    <grounding:wsdlMessagePart rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">
      C:\airReservationService.wsdl#resultFlightDetail
    </grounding:wsdlMessagePart>
  </grounding:Wsd grounding:WsdOutputMessageMap>
</grounding:wsdlOutput>
<grounding:wsdlReference rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">
  http://www.w3.org/TR/2001/NOTE-wsdl-20010315</grounding:wsdlReference>
</grounding:Wsd grounding:WsdAtomicProcessGrounding>

```

Figure 13 Generated Grounding

web services using OWL-S. They recognize the lack of tool support for the development of semantic descriptions. A three step process is introduced in which a tool creates a template using existing software artifacts (e.g. software models, WSDL), identifies relevant ontologies, and performs a classification based on those ontologies. Their tool differs from ours in that it is focused around the use of a matchmaking algorithm to identify relevant ontologies and classify elements in the semantic description with those ontologies. The full specification is then created by mapping concepts in the partial description to those found in the WSDL of existing services. As such, their approach is a bottom up approach that focuses on service description after the fact. Our tool, on the other hand, is a top-down approach that relies on the construction of a semantic service description prior to completing a grounding. Furthermore, it uses a semi-automatic mapping approach to classify elements in the semantic description and relate them to elements in the WSDL.

6. Conclusions and Future Investigations

Semantic Web Services are an evolutionary, if not revolutionary, next step in the advancement of Web service technology. Though in its infancy, the upside of semantic web services far outweigh their downside. One of the primary challenges of using these technologies is one of *population*. That is, there are few semantic web services available for development and experimentation. In this paper we describe a tool intended to overcome the population problem by facilitating the creation of groundings (e.g., mappings of service operations or processes found in OWL-S specifications to concrete and executable realizations of those operations as described by WSDL specifications). As a result, integrating conventional Web services into an OWL-S-based infrastructure can be more simply achieved. The tool is meant to be part of an environment that currently includes an MDA-based tool for writing OWL-S specifications (minus groundings).

In our future investigations we are planning on expanding the capabilities of the grounding tool to more completely support mapping activities including using the tool to assist with creation of operation mediators. Furthermore, we intend to more closely integrate this tool with the OWL-S specification tools that we have previously developed. Finally, we are developing an environment that will allow for the search, discovery, integration and execution of semantic web services.

7. References

[1] The OWL Services Coalition. Owl-s: Semantic markup for web services. [Online] Available <http://www.daml.org/services/owl-s/1.0/owl-s.html>, December 2003.

[2] Joaquin Miller and Jishnu Mukerji et al. MDA Guide Version 1.0.1. Technical Report omg/2003-06-01, Object Management Group, June 2003.

[3] John T. E. Timm and Gerald C. Gannod, A Model-Driven Approach for Specifying Semantic Web Services, in Proceedings of the 2005 IEEE International Conference on Web Services, July 2005.

[4] E. Christensen, F. Curbera, G. Meredith, S. Weerawarana, Web Service Description Language 1.1, W3C Note, 15 March 2001, <http://www.w3.org/TR/wsdl/>

[5] Michael K. Smith, Chris Welty, Deborah L. McGuinness, OWL Web Ontology Language Guide, W3C Recommendation, 10 February 2004. <http://www.w3c.org/TR/owl-guide/>

[6] T. Andrews, F. Curbera, H. Dholakia, Y. Golland, J. Klein, F. Leymann, K. Liu, D. Roller, D. Smith, S. Thatte, I. Trickovic, S. Weerawarana, Business Process Execution Language For Web Services Version 1.1, [Online] Available <ftp://www6.software.ibm.com/software/developer/library/ws-bpel.pdf>, 5 May 2004.

[7] Gentleware, Poseidon for UML, [Online] Available <http://www.gentleware.com/>

[8] Object Technology International, Inc., Eclipse Technology Platform Overview Whitepaper, [Online] Available <http://www.eclipse.org/whitepapers/eclipse-overview.pdf>, February 2003.

[9] Gerald C. Gannod and Sushant Bhatia. Facilitating Automated Search for Web Services. In 2004 IEEE International Conference on Web Services, July 2004.

[10] J. Gennari, M. A. Musen, R. W. Fergerson, W. E. Grosso, M. Crubézy, H. Eriksson, N. F. Noy, S. W. Tu, The Evolution of Protégé: An Environment for Knowledge-Based Systems Development, 2002.

[11] K. Verma, K. Sivashanmugam, A. Sheth, A. Patil, S. Oundhakar, and J. Miller, METEOR-S WSDI: A Scalable Infrastructure of Registries for Semantic Publication and Discovery of Web Services, Journal of Information Technology and Management (under review).

[12] J.Scicluna, C.Abela, and M.Montebello. Visual modeling of owl-s services. In Proceedings of the IADIS International Conference WWW/Internet, October 2004.

[13] Michael C. Jaeger, Lars Engel, and Kurt Geihs. A methodology for developing owl-s descriptions. In First International Conference on Interoperability of Enterprise Software and Applications Workshop on Web Services and Interoperability, February 2005.