CODE: A Development Environment for OWL-S Web services

Naveen Srinivasan Massimo Paolucci Katia Sycara

10/04/2005

CMU-RI-TR-05-48

October 2005

Robotics Institute
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

Abstract

The generation of Semantic Web services is a complex and error prone process and the few tools that are available to support the developer do not form a consistent suite, therefore, they are difficult to use on a consistent basis. CODE, the system that we present here, is an Integrated Development Environment that supports the developer through the whole process from the Java generation, to the compilation of OWL-S descriptions to the deployment and registration with UDDI. In order to achieve this the CODE IDE uses and extends existing web service tools. In this paper we look in detail the architecture CODE IDE and its support through the development and deployment of semantic web services.

Contents

1 Introduction

Web services are becoming an effective paradigm of distributed computation over the Intranet and the all Internet. Furthermore, the interest in Semantic Web services is also growing within the industries, industry standards such as EBXML and UDDI are interested in using OWL in their standards. Within the Semantic Web activity, many important aspects of using semantics within a Web services framework received wide attention. For example, OWL-S [8] stresses the goal directed approach to discovery and composition of Web services, whereas the WSMO framework stresses the importance of mediation. One aspect that has not been analyzed yet is the difficulty of using Semantic Web services languages to describe Web services.

In this paper we analyze this problem from OWL-S view point. The development of OWL-S web services requires many different types of information and activities such as the actual implementation of the Web service; the compilation of the WSDL description; the compilation of the OWL-S Profile, Process Model, and Grounding; the specification of the semantics of all inputs and outputs and their mappings to XML schemata representing the data that flow over the wire. More importantly, the results of all these activities are strictly related: the Profile should represent the capabilities of the Web service, the Process Model should be faithful to the implementation of the Web service, the Grounding should provide a consistent mapping between OWL-S and WSDL, and finally the Web service implementation should be bug free. As a consequence, development of OWL-S web services is time consuming and error prone, and the few tools that are available to support the developers do not form a consistent suite, therefore, they are difficult to use on a consistent basis.

CODE (CMU's OWL-S Development Environment) addresses the problems of the developer by providing a uniform integrated development environment. CODE supports the developer through the whole OWL-S web service development process: from the Java development for web service to the generation of the OWL-S descriptions, to the deployment and registration of the Web service with UDDI. Furthermore, through its OWL-S editors, CODE guarantees the syntactic correctness of the service descriptions, and it allows the developer to use the SPIN model checker [11] to verify correctness claims about the control flow of the OWL-S Process Model. As a result CODE helps the developer to detect problems at development and compilation time, reducing the likelihood of execution time errors.

The guiding principle of the design of CODE is to integrate the tools that the developer needs during the implementation, compilation and deployment of Semantic Web services, in a single consistent and extensible environment. The consistency of the development tools allows the developer to move seamlessly between the different aspects of Semantic Web services development, while the extensibility of the environment allows other parties to use semantic web service framework in their work and to provide additional contributions.

To realize this vision, we implemented CODE as an Eclipse [12] plug-in. Eclipse is known primarily as a Java IDE, but Eclipse goes beyond Java by providing an open platform for plug-in development and for integration of different plug-ins developed by the community. The Eclipse plug-in can be as simple as a source code formatting tools for an existing editor or as complex as a full-fledged Editor. CODE seamlessly inte-

grates Java IDE and other web service frameworks implemented in Eclipse to provide a uniform environment that supports the developer from the design of the system, to its implementation, to its semantic description, to its deployment. Also the contributions of CODE can be integrated with other frameworks developed in Eclipse.

The rest of the paper is organized as follows: Section 2 describes the overview of OWL-S. In section 3, we describe in details the life cycle of the development of semantic web services. In the following section we describe how CODE supports through the development process. In Section 5 we show how CODE also be used by a client to consume the semantic web services generated using the above process. Section 6 illustrates an use case followed by conclusion.

2 OWL-S Overview

OWL-S is a Web Services description language that enriches Web Services descriptions based on WSDL with semantic information from OWL [13] ontologies. OWL-S is organized in three modules: a Profile that describes capabilities of Web Services as well as additional features that help to describe the service; a Process Model that provides a description of the activity of the Web Service provider from which the Web Service requester can derive the interaction; a Grounding that is a description of how abstract information exchanges described in the Process Model is mapped onto actual messages that the provider and the requester exchange.

The OWL-S Profile describes the Web Service capabilities, as well as additional features of Web Services such as provenance and quality of cost specifications of the Web service. The role of the OWL-S Profile is to support different modalities of discovery and to support the requester decision of whether to use a given Web service. OWL-S describes capabilities of Web Services by the transformation that they produce. In addition to capabilities, OWL-S Profiles provides provenance information that describes the entity (person or company) that deployed the service; and non-functional parameters that describe features of the services such as quality rating for the service.

The second module of OWL-S is the Process Model; the Process Model specifies the interaction protocol in the sense that it allows the requester to know what information to send to the provider and what information will be sent by the provider at a given time during the transaction. A Process Model is defined as an ordered collection of processes, where each process produces a state transformation or a data exchange with the Web service clients. The OWL-S Process Model distinguishes between two types of processes: composite processes and atomic processes. Atomic processes correspond to operations that the provider can perform directly. Composite processes are used to describe collections of processes (either atomic or composite) organized on the basis of some control flow structure. For example, a sequence of processes is defined as a composite process of type sequence. Similarly, a conditional statement (or choice as defined in OWL-S) is also a composite process. The OWL-S process model allows any type of control flow structure including loops, sequences, conditionals, non deterministic choice and concurrency. Apart from expressing the control flow between the processes, the process model can also be used to express the data being exchanged between the processes. For example, in a process model of an e-commerce website

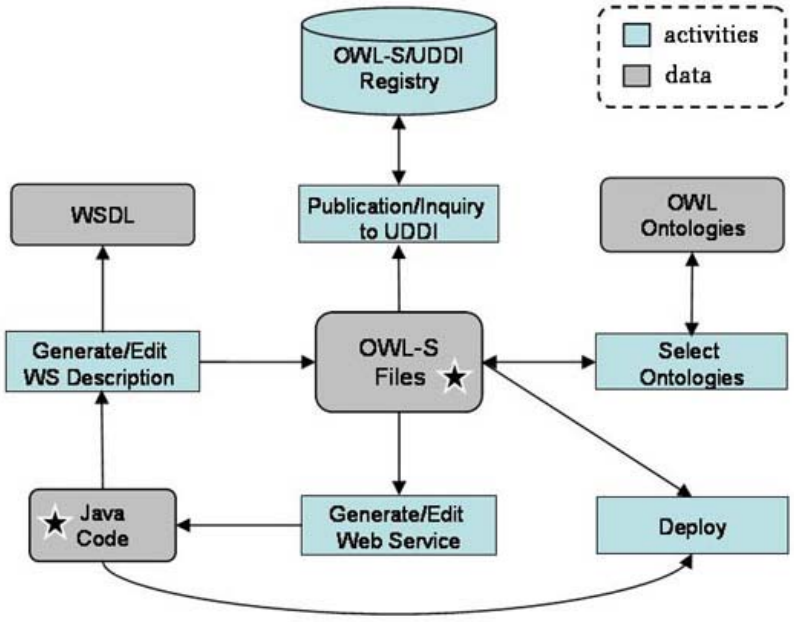

Figure 1: Activities workflow for the generation of OWL-S Web services

the user's identification information that may flow from the login process to other processes may be represented using the dataflow constructs.

The last module of OWL-S is the Grounding that describes how atomic processes; which provide abstract descriptions of the information exchanges with the requesters, are transformed into concrete messages that can be exchanged on the net, or through procedure call. Specifically, the OWL-S Grounding is defined as a one to one mapping from atomic processes to WSDL specifications of messages. From WSDL it inherits the definition of abstract message and binding, while the information that is used to compose the messages is extracted by the execution of the process model.

3 Lifecycle of OWL-S Generation

The development and deployment of a Semantic Web service requires a number of different activities. Figure 1 shows the relation between those activities and what they generate. Crucially there is no clear starting point to the diagram, but two reasonable starting points (indicated by stars in Figure 1) are the Java code and the OWL-S specification. These two starting points define two approached to address the problem of Semantic Web service development. We call the first approach code-driven, and the

second approach model-driven.

3.1 Code-Driven Approach

In the code-driven approach the Web service is implemented using Java or any other programming language, and the OWL-S description is derived from the code. In this approach the developer takes the following steps:

1. develop the Java code to implement a web service;
2. generate the WSDL and OWL-S description of the Web service;
3. use ontologies to enrich the Web service description;
4. publish the OWL-S description of the Web service with UDDI.

This approach naturally fits the development process when we expose legacy systems through Web services. In such a case the functionality is already implemented and a developer only writes code necessary to expose these functionalities as Web services. Crucially, many of these steps can be partially automated to ease the development process and remove the danger of producing erroneous descriptions. For example, in step 2, the generation of the WSDL description is done automatically from the source code of the Web service with web service frameworks such as Axis [1] and .Net [6]; furthermore, the OWL-S description can be partially generated from WSDL with WSDL2OWL-S [2]. Similarly, an OWL-S Profile can be automatically mapped to UDDI advertisements and published in an UDDI using OWL-s2UDDI tool [10].

3.2 Model-Driven Approach

The model-driven approach follows the opposite direction, rather than starting from the Web service code and ending in with the generation of the OWL-S description, the model-driven approach starts with the OWL-S specification of the functionalities required from a Web service, then the specification of the Web service interaction process, and ends with a partial generation of the (Java) code of the Web service. Specifically, the steps in this approach are as follows:

1. the OWL-S description is developed using OWL ontologies;
2. The OWL-S process model is used to implement the Web service;
3. the complete Web service implementation is used to generate WSDL;
4. the OWL-S description is published with UDDI.

The model-driven approach achieves the same level of automation of the code-driven approach, using essentially the same tools. The only exception is step 2, which requires the generation of Java code from OWL-S Process Model descriptions. This generation is done by generating a code-stub for each atomic process specified in the Atomic Process, and by imposing that the processes are executed in the sequence specified by the control flow of the OWL-S Process Model. The result of this process is the

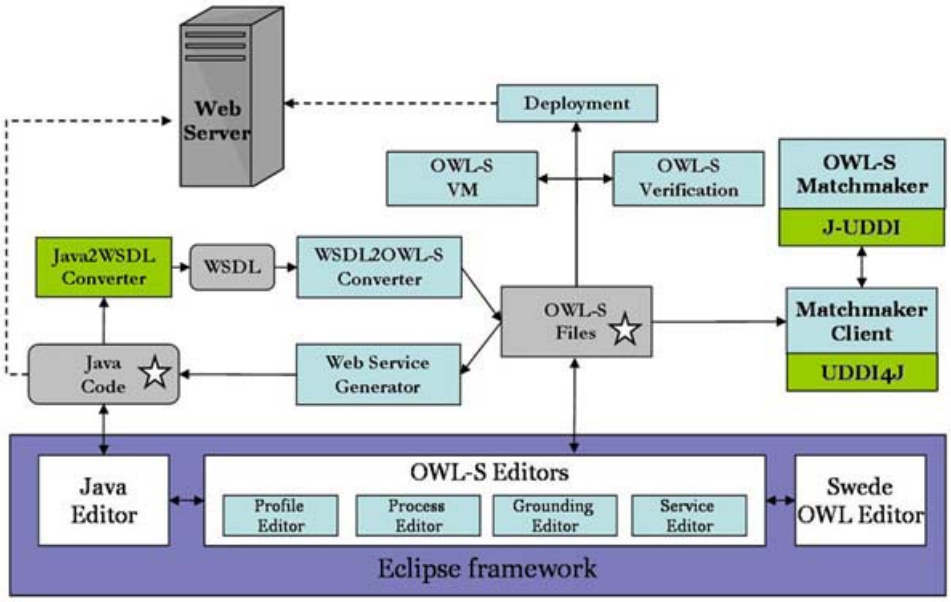

Figure 2: The architecture of CODE

generation of a set of abstract classes that need to be implemented by the Web service developer into concrete classes that perform the functionalities promised by the Web service.

4 CODE

CODE aims at supporting the Web service developer through the whole lifecycle of development of the Web service. In this section we analyze in some details how CODE supports the developer through the lifecycle of an OWL-S service development process.CODE supports both approaches to generate an OWL-S Web service description and leaves the choice to the developer to select the approach that suites their application. CODE supports the Web service developer to perform the following six operations

1. Edit the Web Service code through a close integration with the Eclipse plug-in for Java development;
2. Generate WSDL from the complete the Web service implementation;
3. Generate OWL-S description of the Web service from WSDL;
4. Develop the OWL-S ontology using OWL ontologies;
5. Automatically derive Web service code through he OWL-S Process Model;
6. Publish a Web service description with UDDI;

7. Deploy the Web service publishing the Web service code, WSDL and OWL-S descriptions on a Web server.

Figure 2 shows the architecture of CODE. The main data structure of CODE are, of course the OWL-S Process Model, Profile and Grounding. They are represented internally using the OWL-S API module. The OWL-S API provides Java classes and methods to extract information from an OWL-S description or to generate an OWL-S description. In turn the OWL-S API is based on the Jena [5] OWL models and API. The OWL-S API is the bases for all the other modules that process OWL-S.

Using the architecture is easy to show how these 7 functionalities are achieved. The first functionality, which is the editing of the Java code, is not really part of CODE, but it is provided in the Java plug-in of Eclipse and it is integrated seemly with CODE environment so that the developer does not realize that she is moving between different plug-ins. The second and third functionalities are achieved through the Axis Java2WSDL converter and the WSDL2OWL-S converter. The results of these two steps are a complete WSDL description, and schematic Profile, Process Model and Grounding. Those schematic descriptions contain placeholders for atomic processes, mappings between atomic processes and WSDL operations and placeholders for inputs and outputs in the Profile. But, these descriptions are missing semantic descriptions of the inputs and outputs since WSDL does not provide any semantic description, and they are missing the control flow specification since WSDL does not impose any order on the invocation of operations. The implementation of the code-driven approach results from using these three functionalities.

The fourth functionality is to generate the OWL-S Web service description. This description can be generated either from scratch or by editing an existing OWL-S description generated by the WSDL2OWL-S converter. The three editors at the bottom of Figure 2 provide the editing functionality, with the optional assistance of the SWeDE [9] OWL plug-in for Eclipse. The result of this process is a complete OWL-S description of the Web service that specifies the semantics of all inputs and outputs, the preconditions and effects, and the complete control flow and data flow of the Web service. The fifth functionality is realized by the Web service Generator module. The combination of these last two functionalities supports the model-driven Web service generation.

The last two functionalities: publication with UDDI and deployment are also supported. The publication with UDDI results from the translation of OWL-S Profiles into UDDI Web service descriptions, which are then published into a UDDI registry using a UDDI client. The deployment is supported by packaging the Web service code appropriately and pushing it on a Web Server.

CODE supports two additional functionalities, in addition to the six functionalities reported above. The first one is the Model Checking verification that is implemented as a mapping between the OWL-S Process Model and the Promela language used by the Spin Verifier. Such a mapping allows the developer to verify claims about the Web service. The second functionality is the OWL-S Virtual Machine that is used by CODE to generate automatically the client code.

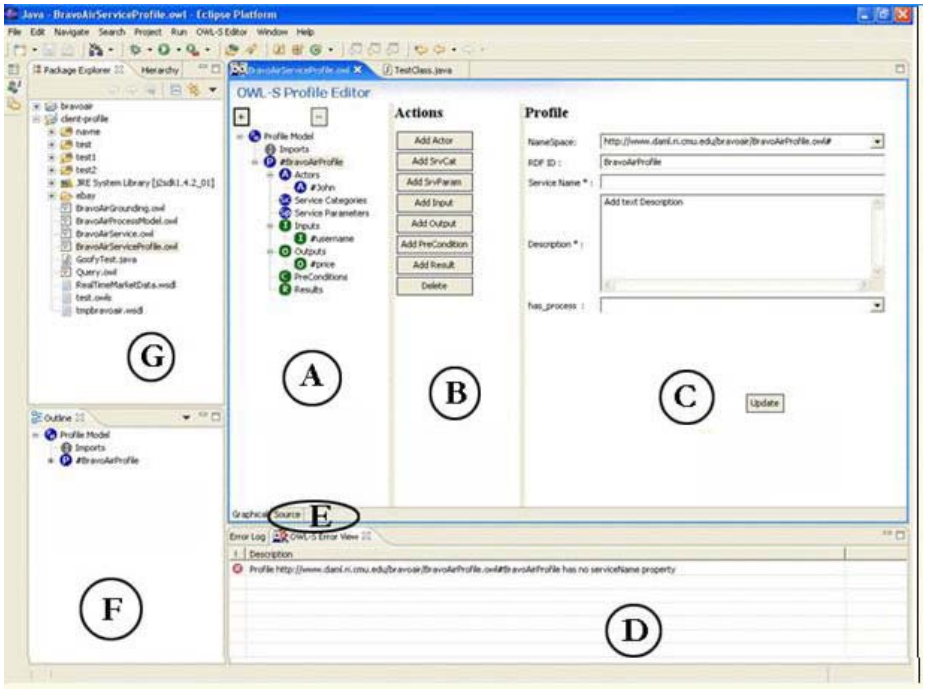

Figure 3: General layout of OWL-S Editors

4.1 General Design of CODE

In this section we will discuss the general design principles observed across all the OWL-S editors in CODE. There are four editors to support different fragments of the OWL-S descriptions namely profile, process model, grounding and service. A uniform user interface is maintained across all four editors.

Figure 3 shows the general layout of the editors. The editor is mainly divided into four parts: file navigation pane, outline pane, main editor pane and error pane. The navigator pane marked as G is a file navigation window and is contributed by the eclipse framework. It is used to browse and manipulate the file system and is also used to manage the projects inside the eclipse workspace. The outline pane is marked as F displays a tree-based synopsis of the file that is being edited. The main editor pane which is composed of area labeled A, B, C and E provides means to edit the OWL-S files. Finally error pane labeled D displays information of about the errors in the OWL-S file that is being edited.

The main editor pane provides two modes editing: form based editing and text based editing. The form editor and the text editor are arranged in a tabular layout stacked one on top of the other, the Figure 3 shows the form editor and the text editor can be accessed using the tab (shown as label E) located at the bottom of the form editor. The form editor provides guidance to the developer on what information should be added at each stage of the compilation of the OWL-S description. For instance,

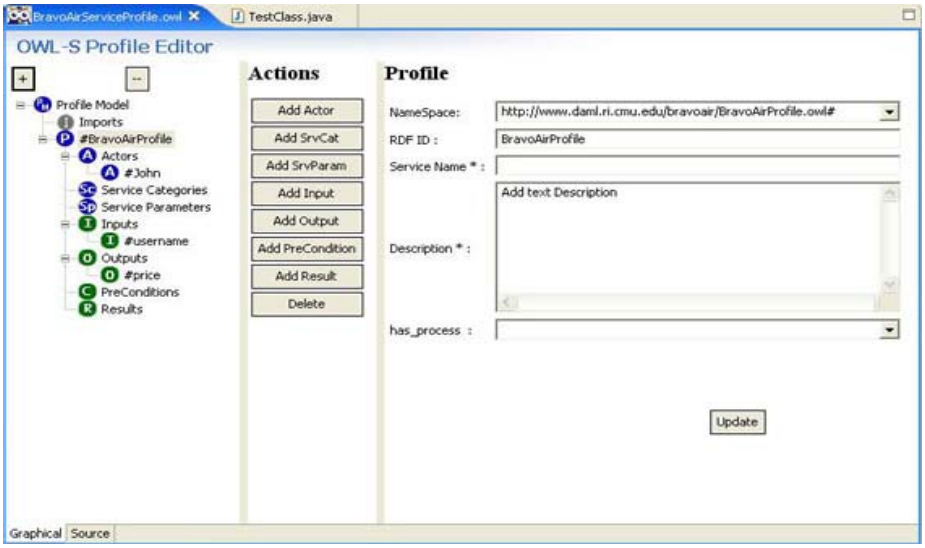

Figure 4: Profile Editor

in the compilation of a process, it requires the developer to enter the inputs, outputs, preconditions and effects. In turn, each one of them is a form that requires the developer to enter the appropriate information. If the information entered is not correct the developer is flagged an error that she can immediately fix.

The form editor is further divided into three sections namely tree pane, action pane and form pane. The tree pane labeled A in Figure 3 presents the elements of the OWL-S description in a hierarchical fashion. The user can browse through the hierarchical structure and may select an element to edit or add an attribute to an element. The action pane and the form pane are marked as B and C respectively are responsive to the selection in the tree pane. The action pane displays the controls such as adding and deleting of attributes that are pertinent to the element selected in the tree pane. Similarly the form pane displays the attributes of the element in form like manner which may be modified.

The text based editor is an extension of SWeDE, an eclipse-based OWL editor. Although this mode of editing is relatively less intuitive than the form editing, it can be used by more experienced developers to develop their OWL-S code more expeditiously as well as to develop ontologies that are used to describe concepts that are specific of the Web service.

Apart from the editing functionalities provided by the OWL-S editors, each editor also offers functionalities specific to the type of OWL-S file it handles. For example the profile editor provides functionalities like publishing, querying etc and likewise the process editor provides functionalities like verification, execution etc. These functionalities that are specific to each editor can be accessed using the drop down menu of the main eclipse window.

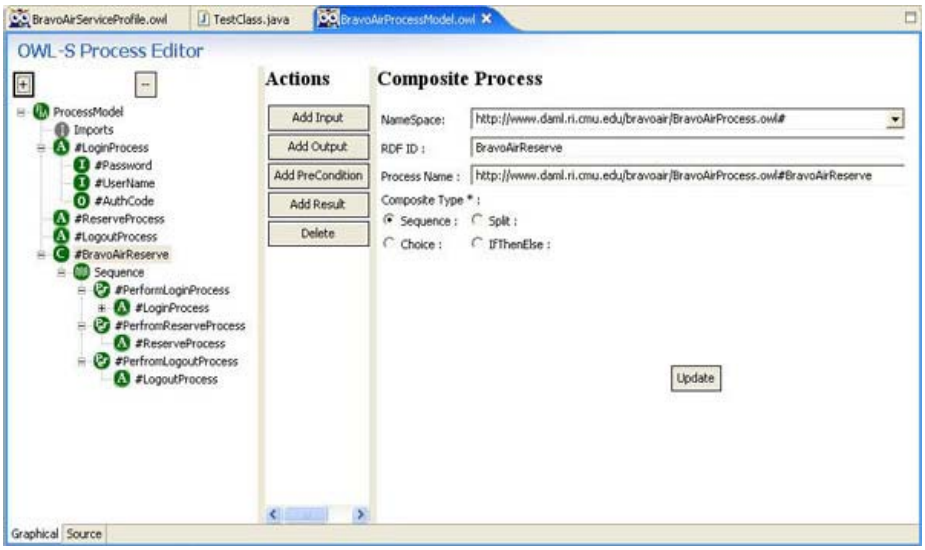

Figure 5: Process Editor

4.2 Profile Editor

The Profile editor supports the developer in the following two tasks: the first one is the editing of the Service Profile of the Web service; the second one is the registration and querying with an UDDI server. Figure 4 displays the form based editor that is used to compile the OWL-S Profile. The leftmost selection of the OWL-S Profile editor shows the hierarchical structure of the profile file that is being edited. The action pane shows the list of actions that can be performed on the node that is selected, in this case a profile is selected in the tree structure. The form pane displays the attributes of the node that is selected.

The profile editor also supports the discovery of semantic web services for both web service developers and web service consumers. On completing the profile a web service developer can register the profile to an UDDI registry. Similarly a consumer look for a web service could compile a profile and use it to query the UDDI registries to look for web services that satisfy her profile. The results returned by the UDDI registry are displayed to the user which may load into the editors for further processing.

4.3 Process Model Editor

The Process Editor supports the developer in the generation of the Process Model using the same approach of the Profile editor. Similar to the profile editor, it provides a form based editor to define processes and their control and data flow. The tree structure in the form editor supports drag and drop operation which expedites the construction of control flow and dataflow in composite processes. When adding a new composite process we add the components of the process which constitutes the control flow of

a process. While display a composite process in process tree its components are displayed in a nested manner and hence one can visualize the control flow of the process model using the structure of the process tree.

A dataflow link to a component of the composite process is added by selecting it and selecting the appropriate action from the action pane. In order to add a dataflow link between two components we need three information, first the name of the input or the output of the component that needs the data, second the name of the other component that generates the data and finally the name of the input or output in the other component that actually has the data.

In addition, the editor provides verification and execution functionalities. The developed Process Model can be verified using the Spin model checker, to eliminate any inconsistencies in the workflow. Likewise, the developer can execute the Process Model using OWL-S VM to eliminate any error during actual invocation of the Web service. The OWL-S VM execution can also be used by client to execute the process model of the service that she discovered using the profile editor.

4.4 Grounding Editor

The grounding editor supports the compilation of grounding descriptions. In order to compile the grounding description the following two files are required: a process model description file containing information about atomic processes and a WSDL file that contain information about the operations and messages exchanged. These two files are loaded into the editor before commencing the compilation process. We generate grounding descriptions by adding information like which process in the process model description maps to which operation in the WSDL file, likewise which input/output of a process maps to which message in the WSDL file. If the user generated the OWL-S description using the WSDL2OWL-S converter, then most of the mapping information will be already present in the grounding file and would be missing very little information. In this case the generated grounding file is loaded into editor to complete the missing information.

4.5 Service Editor

The service editor assist the developer in the compiling the OWL-S service description. The function of the service description is to bind the profile, the process and the grounding descriptions of a web service since there are no explicit links exist between them. Using the service editor we can load the profile, the process and the grounding descriptions and build the service description by choosing the profile, the process and the grounding pair that represents the service.

5 Supporting client

Most of the discussion in the previous sections concentrated on using CODE to support Web service development and description. While this has been the major focus on the service development, CODE can also be used to support the implementation of Web

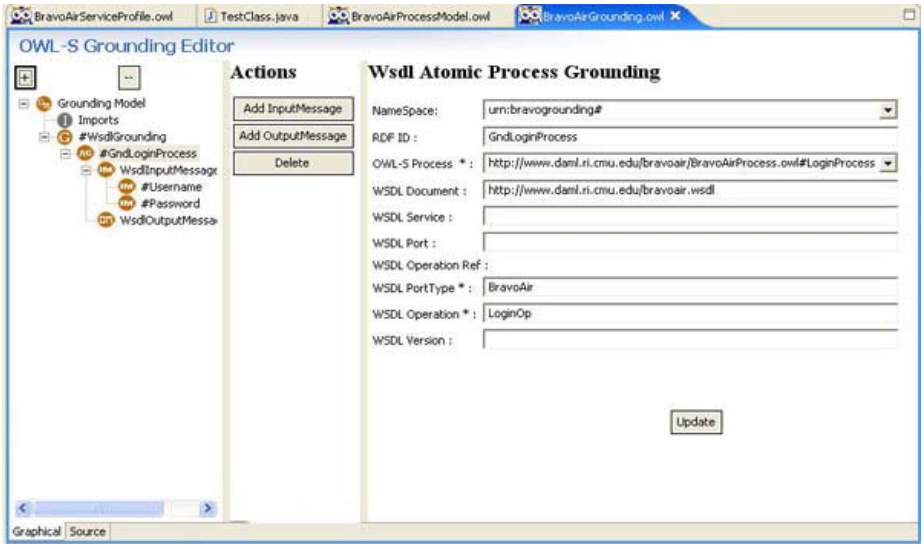

Figure 6: Grounding Editor

services clients. Client development requires two functionalities, first the discovery of Web services that have the capability to solve crucial problems of the client; second the development of the interaction code that allows the client to interact with the Web service.

CODE supports the development of clients on both accounts. The discovery process is support through the interaction with the OWL-S/UDDI. Essentially, to discover a Web service the developer can use the Profile Editor to specify what it expects from the Web service. The developer compiles the profile of the ”perfect” Web service she requires. This profile is translated into a UDDI representation and used to query the OWL-S/UDDI. The result of the query is a set of services that match the query. In some case the match may be perfect, but in most cases, the discovery process will select Web services that are ”similar enough” to the Web service that was originally requested. The client developers use the descriptions of these services to decide which one to use and to decide whether she needs to gather additional information by invoking additional Web services.

The second aspect of the implementation of the client code is the generation of the interaction code with the Web service. The problem here is that the Process Model specifies the order in which the Web service expects information and what type of information the Web service needs. Any violation of such order, or a violation of the type of information expected by the Web service would lead to a failure of the interaction. To control the interaction process, CODE supports the automatic generation of Web service specific interfaces to the OWL-S Virtual Machine. The OWL-S VM takes the OWL-S specification and executes it, but it has to ask to the client what information to send, and which non-deterministic choices to make. By implementing the interface the developer is obliged to provide the functionalities that support the OWL-S VM in its

interaction with the Web service.

6 Conclusions

CODE is an integrated development environment that aims at supporting Web service developers in both the implementation of Web services and in the generation of an OWL-S description of their Web services. The goal of CODE is to support both the server side and the client side developers. The server side developers are supported by providing two modalities of generating Web service descriptions. The first modality is code-driven where the developer generates the OWL-S description directly from the Java code that implements the Web service. The second modality is model-driven where the developer first generates the OWL-S description of the Web services, and then uses it as a model to generate the Java implementation. The client side developers are supported by supporting the discovery of Web services that the client may want to interact with, and by generating automatically the client code that interacts with the Web service.

The functionalities provided by CODE are very similar to the functionalities provided by the OWL-S Editor developed at SRI [3]. The OWL-S Editor is defined as a Protg [7] plug-in and it builds on top of the OWL plug-in already available in Protg. Overall it provides a good support for OWL and a good graphical interface that displays the workflow and data flow of Process Models. At the time of writing is quite difficult to compare the two systems since they are in active development, therefore new functionalities are added all the time. Furthermore, any distinction between the two systems depend more on the underlying support, namely Protg vs Eclipse, rather than profound philosophical differences. A research challenge for the near future will be to analyze the functionalities provided by the two systems to distill a core set of functionalities that are needed to implement Semantic Web services.

Future work on CODE will involve a greater integration with the Eclipse Modeling Framework (EMF) model building tool [4]. EMF allows the developer to move seamlessly between the modeling and the coding of a system. Using EMF, developers may decide how much they want to model the system that they are building and how much they want to develop directly. This approach is very similar to our vision to support both model-driven and code-driven development of Web services. We are currently analyzing how to integrate the development of OWL-S description within the EMF framework.