A Model-Driven Approach for Specifying Semantic Web Services

John T. E. Timm
Dept. of Computer Science & Engineering
Arizona State University - Tempe
Box 878809, Tempe, AZ 85287-8809
area51@asu.edu

Gerald C. Gannod*†
Division of Computing Studies
Arizona State University - East
7001 East Williams Field Road, Mesa AZ 85212
gannod@asu.edu

Abstract

The semantic web promises automated invocation, discovery, and composition of web services by enhancing services with semantic descriptions. One such language used for creating semantic descriptions is the Web Ontology Language or OWL. An upper ontology for web services called OWL-S has been created to provide a mechanism for describing service semantics in a standard, well-defined manner. Unfortunately, the learning curve for semantic-rich description languages such as OWL-S can be steep, especially with given the current state of tool support for the language. This paper describes an automated software tool that uses model-driven architecture (MDA) techniques to generate an OWL-S description of a web service from a UML model. This allows the developer to focus on creating a model of the web service in a standard UML tool, leveraging existing knowledge.

1. Introduction

A *web service* is a loosely coupled component that exposes functionality to a client over the Internet (or an intranet) using web standards such as HTTP, XML, SOAP, WSDL, and UDDI. Of the many challenges of using web services are the problems of specification, search, discovery, selection, composition, and integration. The current state of practice in web services is dominated by the use of the Web Service Description Language (WSDL) [1] to specify access to services. This language lacks an ability to address the aforementioned challenges due to a lack of semantic constructs. 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. Semantically-rich languages such as OWL-S [2] have been created in order to describe concepts and semantics related to a web service using ontologies. Unfortunately, for the common developer, the learning curve for such languages can be steep, providing a barrier to widespread adoption.

Model Driven Architecture (MDA) is an approach to software development that is centered on the creation of models rather than program code. 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 Unified Modeling Language (UML) models.

We are developing an approach that allows a developer to focus on creation of semantic web services and associated OWL-S specifications via the development of a standard UML model. By using an MDA approach, the technique facilitates creation of descriptions of semantic concepts while hiding the syntactic details associated with creating OWL-S specifications. By using transformations from equivalent UML constructs, difficulties caused by a steep learning curve for OWL-S can be mitigated with a language that has a wide user base, thus facilitating adoption of semantic web approaches.

In this paper we describe our efforts to develop a transformation approach for translating UML specifications into equivalent OWL-S specifications. The approach relies upon the use of MDA concepts to translate XMI specifications (e.g., XML encodings of UML) into OWL-S via the use of XSLT transformations [3]. We demonstrate the approach using the CongoBuy example [2], and validate the correctness of our transformations using the Protege ontology specification tool [4].

The remainder of this paper is organized as follows. Section 2 describes background information relevant to our research. The specifics of our approach and the details of a conversion tool are presented in Section 3. Section 4 presents the Congobuy example while Sections 5 and 6 dis-

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

†Contact Author.

cuss related work and conclusions, respectively.

2. Background

This section provides a brief description of web services, ontologies and the XML-based technologies that are referred to throughout the paper.

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. 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.

Web service interfaces are defined using the Web Service Description Language (WSDL) [1]. 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. Ontologies

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 1.

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. Thus far, ontologies have not been widely adopted by software developers.

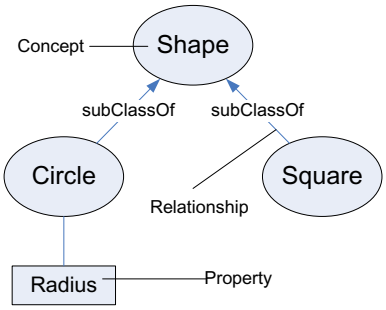

Figure 1. Simple example of an ontology

The Web Ontology Language (OWL) is an XML-based language for describing ontologies [6]. The OWL was designed to allow for the specification of semantic 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 [2]. 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. As such, the OWL-S ontology provides a uniform mechanism for describing the semantics of a web service.

2.3. UML Profiles

A UML profile is a collection of stereotypes, tagged values and custom data types used to extend the capabilities of the UML modeling language. We use a UML profile to model various OWL-S constructs in conjunction with the UML static structure diagram. In terms of MDA, the stereotypes, tagged values, and data types serve to mark-up the platform-independent model, or PIM, in order to facilitate transformation to an OWL-S specification. Stereotypes work well to distinguish different types of classes and create a meta-language on top of the standard UML class modeling constructs. Tagged values allow the developer to attach a set of name/value pairs to the model. A set of name/value pairs is also a convenient way to attach information to the model which is needed in the transformation process.

3. Approach

This section describes our approach for using MDA techniques to synthesize OWL-S specifications.

3.1. Overview

In order for many new techniques and technologies to gain entry into a market, many factors must be considered. While standardization can often be a major reason why adoption of a new technology succeeds, another factor is *ease of use*. That is, if a technology adds little new overhead or requires little in the way of new training, then it is more likely to be adopted. For instance, the benefits of formal methods for software development has been described numerous times [7]. However, an adoption barrier to formal methods by much of the software development community remains in the form of a lack of education. From this standpoint, one of the overarching philosophies of the research described in this paper is the following: *Development of applications that are based on the use of semantic web services should not require knowledge beyond the use of the UML modeling language.* Specifically, it is our belief that in order for techniques that are based on the use of the semantic web and semantic web services, a bridge should be

created using UML that facilitates adoption. In our work, this bridge is enabled via the use of MDA.

Web services have gained a great deal of attention as a way for organizations to use information as a commodity. While many web services currently exist, very few have semantic descriptions associated with them. Furthermore, from the viewpoint of an "end-user", a given web service may or may not provide exactly the data that is needed for a given task. However, it may be the case that a group of web services, when properly composed and integrated, will provide the desired outcomes. Based on the above, we have identified two primary requirements that are being used to drive our approach:

R1 *The approach should be able to incorporate the use of both web services (e.g., services with WSDL specifications only) and semantic web services (e.g., services with semantic descriptions)*

R2 *The approach should facilitate composition of services to form applications or federations.*

The intent of requirement *R1* is to leverage existing web services regardless of the state of their specification. Specifically, the requirement is a recognition of the fact that adoption issues exist in the community and the expectation of widespread use of semantically-rich languages such as OWL-S is unreasonable. However, we do assume that at the very least, a WSDL specification does exist, which is reasonable given that frameworks such as the .NET platform provide tools that automatically generate WSDL.

Requirement *R2* along with the philosophy stated above embody the heart of our approach: a technique for integrating services into composite services by generating an OWL-S specification via the use of MDA.

The OWL-S language consists of three separate parts. The *Service Profile* is a description of what the service does. It is similar to, yet more powerful than WSDL. The *Service Model* is a description of how service works (e.g., the semantics of the service). Finally, The *Grounding Model* is a description of how to access the web service. From a WSDL description, an OWL-S description can be generated using XSLT. However, for semantic web service composition, other information is required to describe the operational behavior of the service (e.g., the service model). We are developing an approach for generating an OWL-S Service Model by taking an XML-based representation of a UML activity diagram and using XSLT to transform the diagram. In this approach, the software developer specifies the operation of a Web service as a composition of other services using UML. An automated design tool then takes the UML description of the service and converts it into the OWL-S Service Model description.

As part of our investigations, we are developing domain models for web services and web service composition as a way to facilitate translation via XSLT of WSDL specifications to OWL-S Grounding Models and UML Activity Diagrams to OWL-S Service Models, respectively. Furthermore, we are investigating the use of different service matchmaking techniques including lightweight signature-based techniques [8] and more computationally expensive semantic techniques [9].

3.2. Framework

Figure 2 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 [10] 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 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 UML tools such as Poseidon [11].

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 will use the profile and process model portions of the OWL-S description as well as a set of WSDL files for corresponding services and automatically generate 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 will be created, as part of the general framework, to automatically generate an executable BPEL specification. 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. The workflow for the macro process can be seen in the diagram below: A semantic Web service composition may have multiple concrete realizations. Completely separate executable specifications can be generated from these multiple groundings.

In the remainder of this paper we discuss the implementation of the first stage of this framework, a conversion tool

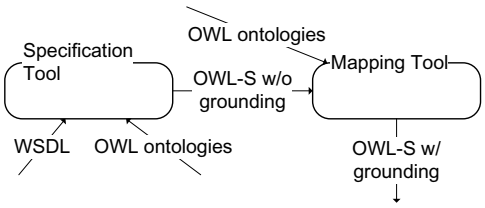

Figure 2. Framework

that synthesizes OWL-S specifications from UML models.

To facilitate the transformation process, we extended the standard UML static structure diagram by creating a UML profile in order to accommodate some of the constructs in the OWL-S specification. The UML profile includes stereotypes, custom data types and tagged values which each map to a particular construct in the OWL-S description as shown in Table 1. The leftmost column provides the abstract type represented by the constructs. The middle column in the table shows the UML constructs that are used to specify semantic services using class diagrams. Finally, the rightmost column names the corresponding target construct in an OWL-S specification. For instance, in a UML class diagram, a UML class combined with a ⟨⟨ Service ⟩⟩ stereotype is used to model a service instance in an OWL-S specification, while a UML association with a ⟨⟨ presents ⟩⟩ stereotype is used to identify a "presents" property in an OWL-S specification.

These UML extensions are used to facilitate the transformation process. The stereotypes help to identify which classes must be generated in the OWL-S description. Tagged values correspond primarily to property values in the output representation. The custom data types are necessary to represent the XMLSchema data types. There are other ways that UML could have been utilized. For example, attributes with default values could have been used instead of tagged values (our current method); however, the latter method is conducive to the transformation process.

In general the transformation rules to map from the UML model to the OWL-S specification are straightforward. The main classes marked with the ⟨⟨Service⟩⟩, ⟨⟨ServiceProfile⟩⟩, ⟨⟨ServiceModel⟩⟩, and ⟨⟨ServiceGrounding⟩⟩ stereotypes each map into a separate output file. These constructs are linked semantically through the stereotyped associations in a diagram (e.g. ⟨⟨presents⟩⟩, ⟨⟨describedBy⟩⟩, and ⟨⟨supports⟩⟩) and are used to generate the respective elements in output documents. Tagged values are used primarily to model properties in the OWL-S specification, especially in the ServiceProfile. Custom data types are used to model XML Schema data types that are used typically used in an OWL-S specification. At present, we allow only one service to be specified per diagram. In future work, we will expand the transformation process to allow for multiple services.

The general architecture of the conversion tool is shown in Figure 3. First, a UML Class Diagram is created in Poseidon [11] with the UML profile for OWL-S. The class diagram is exported in XMI format. The conversion tool is invoked from the command-line and runs multiple transformations on the input file to produce the corresponding Service, ServiceProfile, ServiceModel, and ServiceGrounding OWL-S documents. The transformations come in the form of XSLT transformations automatically performed on an XMI file. From the standpoint of MDA terminology, the process of creating a UML diagram can be considered equivalent to creating a platform independent model (PIM). The XSLT transformations correspond to "Other Information" necessary to generate a second level PIM in the form of an OWL-S specification. The act of specifying groundings, which is part of the framework we are developing, but outside the scope of the tool described here, corresponds to creating a platform specific model (PSM) in the sense that the mapping provides information specific to creating a specific executable implementation of a semantic service.

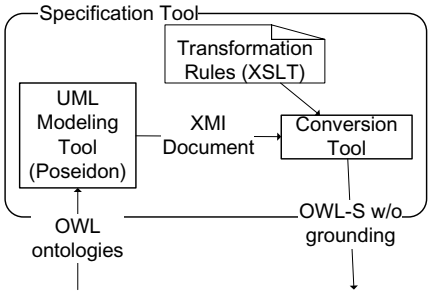

Figure 3. Basic Architecture

3.3. Discussion

The specification conversion tool we have created automatically generates a partial OWL-S specification from a UML model. In its current form, it does not model specific rules used in describing conditional operations as would languages like KIF [12] or SWRL [13]. We are looking at using OCL [14] and transforming from OCL to SWRL in future work.

As mentioned earlier, groundings are not generated by our conversion tool. Instead, we are developing a mapping tool to describing groundings. Given the potential for a wide variety of possible services that could implement a given semantic specification, a specification of a grounding in a UML diagram would make the approach less flexible. Our approach is to generated a stub and to fill in the stub with information from concrete services. To this end, we are developing a mapping tool that takes an abstract service definition and assigns a concrete realization to it by extracting information from concrete WSDL specifications.

Finally, the conversion tool currently is limited to atomic processes. We are expanding the capabilities of the conversion tool to support composition using UML activity dia-

UML Type	UML Construct	OWL-S Construct
Class	<<Service>> stereotype	Service instance
Class	<<ServiceProfile>> stereotype	ServiceProfile instance
Class	<<ServiceModel>> stereotype	ServiceModel instance
Class	<<ServiceGrounding>> stereotype	ServiceGrounding instance
Association	<<presents>> stereotype	"presents" property of Service instance
Association	<<describedBy>> stereotype	"describedBy" property of Service instance
Association	<<supports>> stereotype	"supports" property of Service instance
Parameter	<<ConditionalOutput >> stereotype	ConditionalOutput in ServiceModel
Parameter	<<UnConditionalOutput>> stereotype	UnConditionalOutput in ServiceModel
Data Type	xsd:string	XML Schema data type
Data Type	xsd:integer	XML Schema data type
Data Type	xsd:float	XML Schema data type
Tagged Value	textDescription	textDescription property of ServiceProfile instance
Tagged Value	serviceName	serviceName property of ServiceProfile instance
Tagged Value	actor:name	name property of actor instance
Tagged Value	actor:title	title property of actor instance
Tagged Value	actor:phone	phone property of actor instance
Tagged Value	actor:fax	fax property of actor instance
Tagged Value	actor:email	email property of actor instance
Tagged Value	actor:physicalAddress	physicalAddress property of actor instance
Tagged Value	actor:webURL	webURL property of actor instance
Tagged Value	preCondition	preCondition property of AtomicProcess instance
Tagged Value	coCondition	coCondition property of ConditionalOutput instance
Tagged Value	owl:Class	Standard OWL class

Table 1. UML to OWL Mapping

grams. The combination of tools (conversion and grounding mapper) will generate a full OWL-S specification including OWL-S process models and groundings.

4. Example

The CongoBuy service is an example developed by the OWL-S working group to demonstrate the features of the OWL ontology for services. The example is a fictitious B2C company that offers a number of services including the ability for customers to buy books on the web.

4.1. Specification

Figure 4 shows a class diagram that models the CongoBuy service with the OWL-S stereotypes. Specifically, it shows the specification for an ExpressCongoBuyService that allows a user to directly purchase a book without using a browsing service. The class diagram represents the core structure of the OWL-S description as represented in the UML. Other classes, stereotypes and tagged values are also needed in order to generate the corresponding OWL-S documents.

The example contains seven classes, three of which are OWL classes specified with the ⟨⟨ owl:Class ⟩⟩ UML stereotype. These classes are domain concepts that are relevant to the service being specified. Specifically, they are elements of ontologies pertinent to the book buying and shipping domains and act as "types" for the input and output parameters of the service. The remaining four classes specify the top level service (ExpressCongoBuyService) and its constituent profile, model, and grounding (Profile_Congo_BookBuying_Service, ExpressCongoBuyProcessModel, and CongoBuyGrounding, respectively).

The single operation in the ExpressCongoBuyProcessModel class represents an OWL-S AtomicProcess. Currently, only one Process instance is allowed per ServiceModel instance. Parameters of this operation represent inputs and outputs. Those parameters marked with an "in in the UML model are inputs and those parameters marked with an "out" are outputs. Additionally, output parameters can have a ConditionalOutput stereotype or UnConditionalOutput stereotype. A Parameter with a ConditionalOutput must also have a "coCondition" tagged value, which refers to a condition, which must be true for the output to be generated. Any output parameter that does not have a stereotype is considered to be an UnConditionalOutput. Parameters can either be XML Schema types or they can be standard OWL classes. Any parameters that are not simple data types (e.g., xsd:string, xsd:integer, xsd:float, etc.) must also have a corresponding class in the UML model or be accessible via a UML package namespace. In this example, we directly specify owl:Class stereotype instances. Any number of standard OWL classes can be created and used by the parameters in the AtomicProcess. Tagged values were

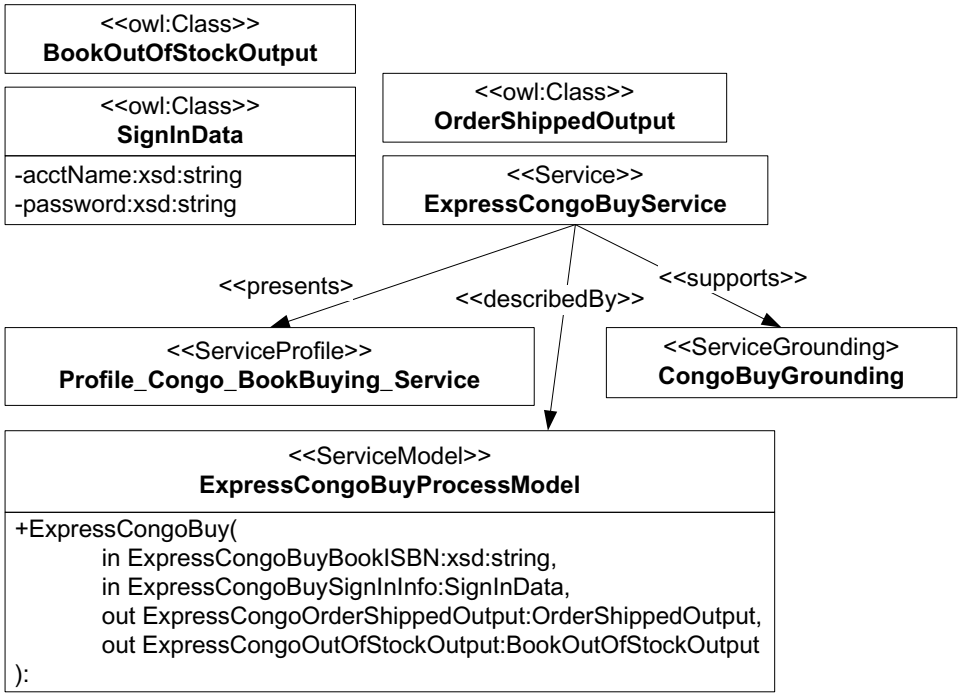

Figure 4. UML with OWL-S Stereotypes

used extensively to annotate the UML model for transformation. Specifically within the ServiceProfile, there were many tagged values used to store the contact information.

Not shown in the diagram are a number of other properties that are specified as part of properties of various classes, a capability provided by many UML-based tools such as Poseidon. These properties are embedded in the resulting XMI representation of the classes. These properties include tagged values added to the various classes which make up the class diagram. The tag definitions and tag values come out in the exported XMI file but are not seen in the class diagram. Tagged values can be manipulated directly through the UML tool. For example, all of the information in the Profile that corresponds to the ⟨profile:contactInformation⟩ element is implemented as a set of tagged values added to the class in the diagram with the ⟨⟨ServiceProfile⟩⟩ stereotype. Also, any preconditions in the process model are implemented as tagged values added to single method in the class marked with the ⟨⟨ServiceModel⟩⟩ stereotype. Finally, any conditional outputs in the process model are implemented as tagged values added to the corresponding method parameters of the single method in the class marked with the ⟨⟨ServiceModel⟩⟩ stereotype.

4.2. Transformations

Figure 5 is an excerpt of an XMI specification of a UML class that would be used as input to the converstion tool. This represents the OWL class ”SignInData”, a parameter type in an OWL-S Process Model presented later. Specifications such as these are transformed using XSLT transformation rules such as the ones shown in Figure 6.

```
<UML:Class xmi.id = '...'
    name = 'SignInData' visibility = 'public'
    isSpecification = 'false' isRoot = 'false'
    isLeaf = 'false' isAbstract = 'false'
    isActive = 'false'>
  <UML:ModelElement.stereotype>
     <UML:Stereotype xmi.idref = '...'/>
  </UML:ModelElement.stereotype>
  <UML:Classifier.feature>
     <UML:Attribute
      xmi.id = 'sm$1620d92:1008b11138c:-7f9d'
         name = 'acctName' visibility = 'private'
         isSpecification = 'false'
         ...>
       ...
     </UML:Attribute>
     <UML:Attribute
      xmi.id = 'sm$1620d92:1008b11138c:-7f9c'
         name = 'password' visibility = 'private'
         isSpecification = 'false'
         ...>
       ...
     </UML:Attribute>
  </UML:Classifier.feature>
</UML:Class>
```

Figure 5. XMI for SignInData class

Figure 6 contains an excerpt of the corresponding XSLT transformation rule for OWL classes. In this particular case, the transformation rule looks for all classes with the owl:Class stereotype and processes them one by one.

An excerpt of a synthesized OWL-S specification is shown in Figure 7. The excerpt shows part of the result

```
<xsl:comment> OWL Classes </xsl:comment>
<xsl:for-each
  select =
  "//UML:Class[*//UML:Stereotype
                [@xmi.idref=$owlClassId]]">
  <xsl:variable name="className"
   select="current()/@name"/>
  <owl:Class rdf:ID="{$className}"/>
  <xsl:if test="current()//UML:Attribute">
    <xsl:for-each select="current()//UML:Attribute">
      <xsl:if test="current()//UML:Class">
        <xsl:variable name="cId"
           select="current()//UML:Class/@xmi.idref"/>
        <xsl:variable name="cName"
           select="//UML:Class[@xmi.id=$cId]/@name"/>
        <owl:ObjectProperty rdf:ID="{@name}">
          <rdfs:domain rdf:resource="#{$className}"/>
          <rdfs:range rdf:resource="#{$cName}"/>
        </owl:ObjectProperty>
      </xsl:if>
      <xsl:if test="current()//UML:DataType">
          ...
          </xsl:choose>
        </xsl:variable>
        <owl:DatatypeProperty rdf:ID="{@name}">
          <rdfs:domain rdf:resource="#{$className}"/>
          <rdfs:range rdf:resource="{$dataType}"/>
        </owl:DatatypeProperty>
      </xsl:if>
    </xsl:for-each>
  </xsl:if>
</xsl:for-each>
```

Figure 6. Excerpt of XSLT Rule

of applying the transformation rules in the conversion tool. The OWL class SignInData refers to a semantic concept that allows for an abstract notion of account name and password to be associated with the concrete structures of account name and password. From the perspective of the user-architect, the entire transformation process is hidden behind the tools, allowing the user-architect to focus on graphical model construction.

```
  <owl:Class rdf:ID="SignInData"/>
  <owl:DatatypeProperty rdf:ID="acctName">
    <rdfs:domain rdf:resource="#SignInData"/>
    <rdfs:range
     rdf:resource="http://.../XMLSchema#string"/>
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:ID="password">
    <rdfs:domain rdf:resource="#SignInData"/>
    <rdfs:range
     rdf:resource="http://.../XMLSchema#string"/>
  </owl:DatatypeProperty>
```

Figure 7. Excerpt of Synthesized OWL-S

4.3. Discussion

In the current version of our tools, as stated previously, we specify atomic processes. We are currently developing an approach for supporting composite services in order to specify more complex services.

Since there may be many possible groundings for any given semantic web service, the specification of the service grounding is left as an additional operation that falls outside of the scope of the specifications that our tools synthesize. Instead, we are developing tools that allow a user to specify groundings by creating mappings to services specified with the web services description language, WSDL.

In order to identify any syntactic or semantic errors in generated OWL-S specifications documents are loaded into the Protege tool, which validates the OWL-S ontologies. Protege allows for further editing of the OWL-S ontology and other similar manipulation operations, but for our purposes it primarily serves as a validation tool for generated OWL-S descriptions.

Our experiences with this approach have demonstrated that the difficulties associated with generating OWL-S specifications can be isolated to making sure the UML specifications are complete, an operation that can easily be verified by analyzing synthesized OWL-S specifications with Protege. Our future investigations include performing a user study to determine whether our technique provides an overall savings in time in writing OWL-S specifications when compared with writing OWL-S specifications directly and within other tools such as Protege.

5. Related Work

Currently, there are only a few tools that support visual development of OWL- S descriptions. These tools are outlined below. Protege is a software tool, which provides a visual environment for editing ontologies and knowledge bases [4]. The OWL plug-in, for Protege, allows for visual editing of OWL-based ontologies but does not specifically support the OWL-S ontology for services. Because the tool supports generic OWL ontologies, there is no mechanism to visualize OWL-S processes. Therefore, creating composite processes is difficult to achieve in a tool like Protege. Protege works well for general ontology development but lacks features required to model specific OWL-S constructs and demands its own set of techniques. The learning curve for such a tool can be steep.

Elenius et al. have developed an OWL-S plug-in for Protege [15]. The OWL-S plug-in allows for the modeling of semantic web services within the Protege environment. The plug-in will also transform a WSDL specification into a partial OWL-S specification which can then be manipulated in a visual environment. Our tool focuses on the forward engineering of the semantic web service and then requires a mapping tool to ground the semantic web service to a concrete service realization. Furthermore, the goal of our research is to create tools that do not require any knowledge beyond UML to develop semantic web services and thus avoids use of environments like Protege.

The OWL-S Editor supports visual editing of OWL-S descriptions [16]. It provides a graphical user interface to create and modify an OWL-S description including all three parts: Service Profile, Service Model, and Service Grounding. The Visual Composer feature of the editor allows

for the modeling of a composite process using a standard UML Activity Diagram. The OWL-S Editor also provides a mapping mechanism between WSDL and OWL-S. Our approach leverages existing skills in UML modeling which can greatly improve the efficiency of the semantic web service development workflow.

Jaeger, Engel and Geihs [17] have developed a three step process for generating OWL-S specifications by create a template using existing software artifacts (e.g. software models, WSDL), automating the identification of relevant ontologies, and peforming a classification based on those ontologies. Their methodology differs from ours in that it is focused around the use of a matchmaking algorithm to identify relevant ontologies. Elements are then classified in a a semantic description using those ontologies. Our work focuses on the creation of models, leveraging existing knowledge of UML, and generating a partial OWL-S specification from those models. The full specification is then created by mapping concepts in the partial description to those found in the WSDL of existing services.

6. Conclusions and Future Investigations

OWL-S provides an ontology for web services, which can be used to describe the semantics of a web service. Unfortunately, adopting something like OWL-S can be difficult because of the learning curve and current state of tool support. We have create a UML profile for OWL-S in order to allow software developers to model OWL-S descriptions with a standard UML modeling tool. The conversion tool described in this paper was created to take an XML representation of the UML model and convert it into an OWL-S description. Some issues that will be resolved in future work include: modeling effects for atomic processes, modeling composite processes using a behavioral diagram (e.g. UML Activity Diagram) and a tool to allow for mapping between a WSDL file and concepts in an OWL-S service model as a means for specifying an OWL-S service grounding.

The research issue of particular interest is that of automated composition of services. Currently service composition is performed at design-time. With semantic descriptions in place and mappings between the descriptions and the concrete realizations of the services, automated composition is possible. As part of our research, we are investigating how the use of a product-line approach that focuses on commonalities and variabilities can be used to develop OWL-S service specifications as commonalities and OWL-S service groundings as variabilities.

References

[1] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana. Web service description language 1.1. W3C Note [Online] Available http://www.w3.org/TR/wsdl/, 2001.

[2] OWL Services Coalition. Owl-s: Semantic markup for web services. [Online] Available http://www.daml.org/services/owls/1.0/owl-s.html, Dec. 2003.

[3] J. Clark. XSL Transformations v1.0. W3C Recommendation [Online] Available http://www.w3c.org/TR/xslt, Nov. 1999.

[4] J. Gennari, M. A. Musen, R. W. Fergerson, W. E. Grosso, M. Crubezy, H. Eriksson, N. F. Noy, and S. W. Tu. The evolution of protege: An environment for knowledge-based systems development. Unknown, 2002.

[5] Tim Berners-Lee, James Hendler, and Ora Lassila. The semantic web. *Scientific American*, May 2001.

[6] Michael K. Smith, Chris Welty, and Deborah L. McGuinness. Owl web ontology language guide. W3C Recommendation [Online] Available http://www.w3c.org/TR/owl-guide/, February 2004.

[7] E. W. Clarke and J. M. Wing. Formal Methods: State of the Art and Future Directions. Technical Report CMU-CS-96-178, Carnegie Mellon Univ., August 1996.

[8] Gerald C. Gannod and Sushant Bhatia. Facilitating automated search for web services. In *Proceedings of the 2004 IEEE International Conference on Web Services*, July 2004.

[9] K. Sycara, M. Paolucci, A. Ankolekar, and N. Srinivasan. Automated discovery, interaction and composotion of semantic web services. *Journal of Web Semantics*, 1(1), 2003.

[10] T. Andrews, F. Curbera, H. Dholakia, Y. Goland, J. Klein, F. Leymann, K. Liu, D. Roller, D. Smith, S. Thatte, I. Trickovic, and S. Weerawarana. Specification: Business process execution language for web services v1.1. [Online] Available http://www- 128.ibm.com/developerworks/library/ws-bpel/, May 2003.

[11] Gentleware. Poseidon for uml. [Online] Available http://www.gentleware.com/.

[12] Michael R. Genesereth and Richard Fikes. Knowledge interchange format (KIF). draft proposed American National Standard, NCITS.T2/98-004, 1998.

[13] I. Horrocks, P. Patel-Scheider, H. Boley, S. Tabet, B. Groshof, and M. Dean. SWRL: A Semantic Web Rule Language Combining OWL and RuleML. DARPA DAML Program, 2003.

[14] OMG. Object constraint language specification. OMG Unified Modeling Language Specification, V1.3, June 1999.

[15] D. Elenius, G. Denker, D. Martin, F. Gilham, J. Khouri, S. Sadaati, and R. Senanayake. The owl-s editor - a development tool for semantic web services. In *Proceedings of the Second European Semantic Web Conference*, May 2005.

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

[17] 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.