

Specifying Semantic Web Service Compositions using UML and OCL

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*†
Dept. of Comp. Science and Systems Analysis
Miami University
123 Kreger Hall, Oxford, OH 45051
gannodg@muohio.edu

Abstract

The semantic web promises to bring automation to the areas of web service discovery, composition and invocation. In order to realize these benefits, rich semantic descriptions of web services must be created by the software developer. A steep learning curve and lack of tool support for developing such descriptions thus far have created significant adoption barriers for semantic web service technologies. In this paper, we present a model-driven architecture based approach for specifying semantic web service compositions through the use of a UML profile that extends class and activity diagrams. This profile is used in transformations that facilitate automatic construction of OWL-S specifications from UML diagrams. Conditions required by the composition, such as those on control constructs, are specified using OCL and transformed into SWRL during the construction process.

1. Introduction

The *semantic web* is considered by many to be the future of the current web [1]. Resources in the semantic web are enhanced (i.e., given semantics) using rich description languages such as the Web Ontology Language (OWL) [2]. Intelligent software agents can, in turn, use these descriptions to reason about web resources and automate their use to accomplish goals specified by the end-user including intelligent search and retrieval. Creating semantic descriptions requires skill in the areas of knowledge engineering and representation.

Semantic web services are web services that have been enhanced with formal semantic descriptions. Semantic descriptions as expressed with ontologies and description logics enable automated discovery, composition and invocation of services. In this context, an *ontology* is a formal conceptualization of a particular domain. A description is created

using concepts from that domain, properties of those concepts, and relationships between concepts. In this way, ontologies are useful for describing the semantics of a web service. As such, developers must create ontologies to describe service capabilities, inputs, outputs and execution semantics - a task that many are unfamiliar with. Subsequently, to facilitate adoption, it is important to create tools that can leverage developers' experiences in data and object-oriented modeling to create ontologies that can be used to describe web services.

Web service compositions described using semantic technologies provide several advantages over current syntax-based technologies such as BPEL [3]. Semantic descriptions of compositions can be given to reasoners for more intelligent service discovery and matchmaking. Semantic rules can be used to provide pre and post conditions for service compositions. These pre and post conditions can be evaluated at runtime. Furthermore, semantic web service compositions describe services at a higher level of abstraction, which facilitates the development of dynamic service-binding techniques.

In this paper, we describe a methodology grounded in the use of model-driven architecture for specifying semantic web services. In previous work, we focused on specification of *atomic processes* in OWL-S [4], a semantic web service language. This paper addresses *composition operators* by facilitating the generation of OWL-S using UML as a specification language and XSLT transforms for synthesizing descriptions of composite processes. The paper also addresses the use of OCL [5] to specify various conditions required by the composition including pre and post conditions on processes and conditions of control constructs. In our approach, OCL conditions are converted into SWRL [6] using additional XSLT transforms. There are several challenges involved in creating semantic descriptions of service compositions. The first challenge is a syntactic one. Developers must learn new languages such as OWL and SWRL in order to create semantic descriptions. Current tools create a steep learning curve and the XML-based syntax itself

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

†Contact Author.

can be verbose and error prone. Developers must also learn the semantics of these description languages. The UML is suitable for use as an ontology language [7, 8]. The default semantics of the UML class and activity diagrams are similar to those of a semantic description language. In those instances where UML has semantic differences, standard UML extensions such as stereotypes and tagged values are provided. As such, our approach addresses both the syntactic and semantic challenges noted above.

The remainder of this paper is organized as follows. Section 2 describes background material relevant to our research. Section 3 presents our framework and introduces our approach for constructing OWL-S compositions using UML including transformation of OCL into SWRL. An example that illustrates the approach is provided in Section 4, while Section 5 describes related work. Finally, Section 6 draws conclusions and discusses future investigations.

2. Background

Ontologies and Semantic Web Services. 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 [1]. Ontologies serve as the primary building block for adding semantics to web services in this project. Modeling with ontologies is similar to domain modeling in software engineering with analogies between the two disciplines including the use of classes, inheritance, and attributes.

The Web Ontology Language (OWL) is an XML-based language for describing ontologies [2]. The OWL was designed to allow for the specification of semantic descriptions for resources on the Web, facilitating interpretation by autonomous software agents. These resources may be anything from simple web pages to complex web services. Because OWL descriptions have an XML-based syntax, they are platform-independent and can easily be transferred over a network and manipulated with existing tools.

OWL-S [4] is an upper ontology for semantic web services written in OWL. The metamodel for the OWL-S ontology is depicted in Figure 1 using a UML diagram. The OWL-S ontology is intended to provide a base for creating semantic descriptions of services and is divided into three main parts. The *ServiceProfile* describes what the service does and is used for service discovery. The *ServiceModel* describes how the service works and is used for service composition. A service composition may contain both atomic and composite processes. An atomic process represents an indivisible unit of computation while a composite process is made up of control constructs and other processes (either *atomic* or *composite*). The *ServiceGrounding* describes how to access and use concrete services.

The Semantic Web Rule Language (SWRL) is used to express rules that extend the axiomatic capabilities of

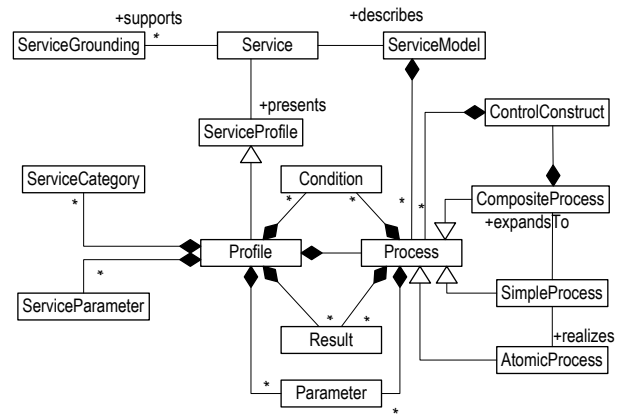


Figure 1. OWL-S Metamodel in UML

OWL [6]. These rules can be used to express conditions for OWL-S specifications including pre/post-conditions and effects. SWRL rules can also be used to express conditions in control constructs of OWL-S composite processes.

UML and OCL. The Unified Modeling Language (UML) is a general-purpose modeling language for software systems [9]. The language can be used to model all static and dynamic aspects of the system including both structure and behavior. The UML uses a graphical notation to illustrate various aspects of a software system, and can also be serialized into an XML-based textual format for interoperability between modeling tools. The semantics of the UML can be extended using profiles to fit modeling into a particular domain such as semantic web services. UML profiles can contain *stereotypes* and *tagged values*. Stereotypes are names that, when attached to model elements, are used to convey the meaning of that element. Tagged values are name/value pairs which are attached to model elements and provide a mechanism for supplying additional information about the element. UML has widespread tool support and has been widely adopted in both industry and academia.

The Object Constraint Language (OCL) is a language used for representing expressions for UML models [5]. Typically, OCL expressions are used to represent invariants that hold true over a UML model. One important aspect of OCL conditions is that they do not have side effects. Evaluating an OCL expression will not modify the state of a model. Evaluating an expression, however, may call an operation that does change the state of the model.

3. Approach

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.

We are developing an approach that lessens the burden of specifying semantic web services on software developers already familiar with UML and OCL. In our approach, the software developer specifies a web service at a higher level of abstraction than that of WSDL. Specifically, in our approach, the structure of the service is specified using a UML Class Diagram, pre and post conditions on service operations are specified using OCL, UML Activity Diagrams are used to specify service compositions, and OCL is also used to specify conditions required by service compositions. An automated design tool then takes the UML model of the service and transforms it into an OWL-S specification. Any conditions specified using OCL are transformed into SWRL expressions. As part of our investigations, we are developing domain models for web services and web service compositions as a way to facilitate the creation of OWL-S Grounding Models using WSDL specifications. Furthermore, we are investigating the use of different service matchmaking techniques including lightweight signature-based techniques [10] and more computationally expensive semantic techniques [11].

3.1. Framework

This research focuses on the creation of a model-driven framework for developing semantic web services. The framework supports the specification of semantic web services, the grounding of those services to concrete service implementations and the execution of those services. Forward-engineering is employed when using the framework to develop semantic web service descriptions. Models are the primary artifact of the development process. Thus, any changes that happen at the model level require service descriptions to be regenerated. Consequently, manual changes made to the generated descriptions are not propagated backwards.

The framework relies on a set of software tools that were created to support specification and execution. The framework architecture is seen in Figure 2.

The specification tool transforms UML models, created with a UML Profile for OWL-S, into a partial OWL-S specification. Because both the UML models and the OWL-S specifications use an XML-based serialization format, XSLT has been chosen as the transformation language.

The following transformations have been created: transformation of stereotyped UML classes to instances of classes in the OWL-S ontology, transformation of stereotyped UML classes to standard OWL classes, transformation of stereotyped UML packages to represent the dependence on external ontologies and the transformation of stereotyped UML activity diagrams to an instance of the OWL-S composite process class, including control constructs such as *If-Then-Else*, *Split-Join*, etc.

Utilizing a graphical user interface, the grounding tool

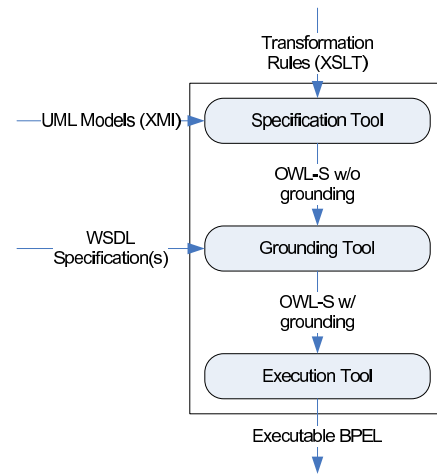


Figure 2. Framework

allows the developer to map concepts in the partial OWL-S specification to input/output messages and operations in a WSDL file. The grounding tool uses this mapping to generate the full grounding for the OWL-S specification. The execution tool uses the OWL-S API [12] to execute the grounded specification.

3.2. Specification of Atomic Processes

In previous work, we developed an approach for generating atomic process specifications in OWL-S using UML [13]. In that work, we created a UML profile for semantic web services that included definition of stereotypes and tagged values for facilitating both the specification task as well as the transformation task. An excerpt of the UML profile that we have defined for atomic processes is shown in Table 1(a). The first column contains the UML extension used (e.g. stereotype, tagged value, etc.). The second column contains the UML model element to which the extension is applied. The third column contains the OWL-S construct that is represented by the extended UML model element. The primary specification vehicle that we used for atomic processes was the UML class diagram.

3.3. Specification of Composite Processes

Individual web services can be composed to create complex composite behaviors and workflows. A composition, in turn, can itself be exposed as a web service. As a result, services can be reused in multiple, different compositions. Those compositions can be static (i.e., determined before runtime) or, they can be dynamic (i.e., determined at runtime). A BPEL [3] orchestration is an example of a static composition. If a web service has been described semantically using concepts from an ontology, it is possible for a software agent or reasoner to create compositions using that service dynamically at runtime. It is also possible to choose which services to bind to using a semantic description of the composition and plugging in concrete services that are required for a specific functionality [13]. Composite pro-

UML Extension	Model Element	OWL-S Construct
<<presents>>	Association	Service instance presents property
<<describedBy>>	Association	Service instance describedBy property
<<supports>>	Association	Service instance supports property
<<Service>>	Class	Service instance
<<Profile>>	Class	Profile instance
<<AtomicProcess>>	Class	AtomicProcess instance
<<Grounding>>	Class	Grounding instance
<<owl:Class>>	Class	inline OWL class definition
<<owl:Ontology>>	Package	external ontology
hasPrecondition	Operation	Process instance hasPrecondition property
inCondition	Operation	Process instance inCondition property
hasEffect	Operation	Process instance hasEffect property

(a) Atomic Processes (Excerpt)

UML Extension	Model Element	OWL-S Construct
<<then>>	ActivityEdge	If-Then-Else control construct instance then property
<<else>>	ActivityEdge	If-Then-Else control construct instance else property
<<CompositeProcess>>	CallAction	CompositeProcess instance
<<If-Then-Else>>	DecisionNode	If-Then-Else control construct instance
<<Repeat-While>>	DecisionNode	Repeat-While control construct instance
<<Split-Join>>	ForkNode	Split-Join control construct instance
ifCondition	DecisionNode	If-Then-Else control construct instance
whileCondition	DecisionNode	Repeat-While control construct instance whileCondition property

(b) Composite Processes (Excerpt)

Table 1. UML Profile for OWL-S

cesses differ from atomic processes in that they maintain state. The state of a composite process is changed each time a message is sent to it by the client [4].

The specification of composite processes involves additional complexity beyond that of atomic processes. While atomic processes are the building blocks for composite processes, control constructs are used to “glue” together atomic processes into composite processes. These control constructs, as described in Section 3.4, also require the specification of conditions. Additional complexity comes from the requirement to support data binding between processes in a composition. Certain processes may require complex compositions that involve many atomic processes and even other composite processes. In order to support these types of complex compositions, a hierarchical approach is required. Using the UML Activity Diagram patterns in Section 3.3.2, the developer creates complex compositions by nesting these simpler patterns inside one another creating one activity diagram for each activity with the <<CompositeProcess>> stereotype.

3.3.1. Structural Concerns

Figure 3 shows a class diagram that models a semantic web service. Class diagrams used to model composite processes contain several features that facilitate the specification of service compositions. An operation with the <<Atomic-Process>> stereotype is transformed directly into an OWL-S *AtomicProcess* construct. An operation with the <<CompositeProcess>> stereotype indicates that there is an obligation to specify a corresponding UML Activity Dia-

gram.

To reference classes in external ontologies a UML package is specified with the stereotype <<owl:Ontology>> and the tagged value *uriRef*. For each class that will be used in the model, a placeholder class must be created so that the class can be referenced in the model.

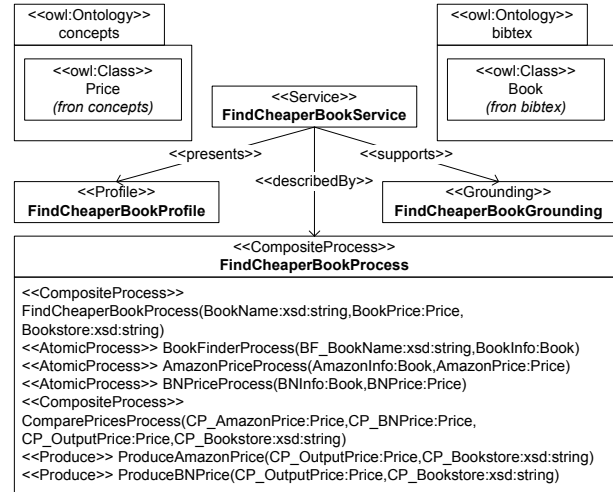


Figure 3. UML class diagram for example

3.3.2. Composition Using UML Activity Diagrams

Mappings from UML to OWL-S for specifying composite processes have been added to our previous model-driven framework for semantic web service development [13]. The mappings encapsulate the relationships between model elements in the UML activity diagram and concepts/properties in the *ServiceModel* of the OWL-S ontology. As such, composite processes in the OWL-S *ServiceModel* can use any of the following OWL-S control constructs: *Sequence*, *If-Then-Else*, *Split-Join*, *Choice*, *Any-Order*, *Repeat-While* and *Repeat-Until*. The mappings also support the description of the flow of data between the output of one service and the input of another service.

The additional UML profile extensions for composite processes are shown in Table 1(b). Semantic extensions for modeling elements in the UML activity diagram have been added to the profile including: stereotypes on call actions, decision nodes, merge nodes, fork nodes and join nodes and tagged values. The stereotypes identify instances of control constructs in the *ServiceModel* of the OWL-S ontology.

Additional tagged values, not shown here, are used for data binding between processes in the composition. The *hasDataFrom* tagged value (not shown) is used to specify how a parameter for a particular process binds to a parameter in another process. The *ifCondition* tagged value represents the condition which must be true in order to execute the “then” path in the *If-Then-Else* construct. In a UML activity diagram, the “then” path is identified with the <<then>>

stereotype. Similarly, the “else” path is marked, in the diagram, with the <<else>> stereotype.

Figure 4 contains a subset of the patterns of UML activity diagrams that are used to model equivalent OWL-S composition constructs. Activity diagrams use a graphical notation similar to flowcharts, where activities are represented using boxes with rounded corners and decision structures are represented using diamonds. In the diagrams, the activities marked with the <<AtomicProcess>> stereotypes can be alternatively stereotyped with <<CompositeProcess>>. This allows for more complex compositions to be specified. The *Sequence* construct uses a straightforward representation. A *Sequence* is a series of processes which execute one after another. The semantics of UML activity diagrams matches the OWL-S *Sequence* construct. An example of the *Sequence* construct in UML is seen in Figure 4(a). The *If-Then-Else* construct comes in two versions. The first version has two paths of execution: a “then” path and an “else” path and is seen in Figure 4(b). The second version (not shown here) has no “else” path and translates to the conditional execution of one atomic or composite process.

The *Split-Join* construct represents concurrent execution of multiple processes. It closely follows the parallel activity construct in UML activity diagrams. The *Split-Join* construct was also straightforward to implement. An example of the *Split-Join* construct is seen in Figure 4(c). The *Choice* construct is similar in structure to *If-Then-Else* in terms of UML implementation. A *Choice* construct specifies that any path can be chosen and executed from a given set of paths. The *Any-Order* construct is modeled similarly to a *Split-Join*. An *Any-Order* construct specifies that processes are executed in any order but not concurrently. Although the graphical notation is the same, it is the stereotype that makes differentiates between the semantics of the two constructs. The two looping constructs *Repeat-While* and *Repeat-Until* are modeled using single-entry, single-exit design patterns. They are represented by using analogous UML activity merge and decision nodes. The primary difference between the two looping mechanisms is the control flow. *Repeat-While* is depicted in Figure 4(d). The *Repeat-While* construct evaluates the *whileCondition* before it conditionally executes the *whileProcess*. The *Repeat-Until* construct executes the *untilProcess* first and then checks the *untilCondition* before continuing. The semantics of these constructs are similar to the while statement and do-while statement in a general programming language.

In order to support composite service specifications, additional transformation rules are required above those we have developed for atomic processes [13]. These rules are based on the additions to the UML Profile for OWL-S. Because composite processes can contain other composite processes, a recursive approach was used in creating the rules for composite processes. An excerpt of the transformation

rule for the *If-Then-Else* construct is seen in Figure 5.

```
<process:If-Then-Else>
  <xsl:choose>
    <xsl:when test="contains($condition, ':')">
      <process:ifCondition>
        <expr:SWRL-Condition rdf:ID="...">
          <expr:expressionBody rdf:parseType="Literal">
            <xsl:call-template name="ocl2swrl">
              <xsl:with-param name="expression" select="...">
              <xsl:with-param name="operationName" select="...">
            </xsl:call-template>
          </expr:expressionBody>
        </expr:SWRL-Condition>
      </process:ifCondition>
    </xsl:when>
    <xsl:otherwise>
      <process:ifCondition rdf:resource="#{ $condition }">
        <xsl:otherwise>
        </xsl:otherwise>
      </process:ifCondition>
    </xsl:choose>
    <process:then>
      <xsl:call-template name="handleNode">
        <xsl:with-param name="node" select="$thenNode"/>
      </xsl:call-template>
    </process:then>
    <xsl:if test="name($elseNode)='UML2:CallAction'">
      <process:else>
        <xsl:call-template name="handleNode">
          <xsl:with-param name="node" select="$elseNode"/>
        </xsl:call-template>
      </process:else>
    </xsl:if>
  </process:If-Then-Else>
```

Figure 5. If-Then-Else Rule (Excerpt)

3.4. Translation of OCL to SWRL

To support complex activities such as automated reasoning and matchmaking, semantic information needs to be provided in the control specifications described above. Through the use of OCL and UML activity diagrams, the SWRL specifications generated by our approach capture a portion of the behavioral semantics of the web service composition (e.g. preconditions, postconditions, and conditions on control constructs).

In order to support all of the information required to make the transformation between UML/OCL and OWL-S/SWRL, a well-defined tagged value syntax was created (see Table 2). In the table, the tag definition used on the model element is seen in the first column. The second column contains a simple BNF syntax for the expression. The syntax provides a standard mechanism for specifying OCL expressions and data binding information in a UML model. The syntax consists of comma separated lists, with OCL expressions and their names being separated using a colon. The expression may be omitted when the name refers to an expression that has been defined elsewhere in the model. This provides a convenient way to reuse expressions.

The *hasPrecondition*, *inCondition*, and *hasEffect* tag definitions are used to specify various conditions and expressions. The *hasLocal* and *hasResultVar* are used to declare variables that are used in these expressions. The *hasDataFrom*, *withOutput*, and *producedOutput* are tag definitions used to specify the input and output bindings used in data flows between processes. All of the tags are used as extensions to a UML operation which is part of the process class. The *producedOutput* tag definition is used in conjunction with the <<Produce>> stereotype.

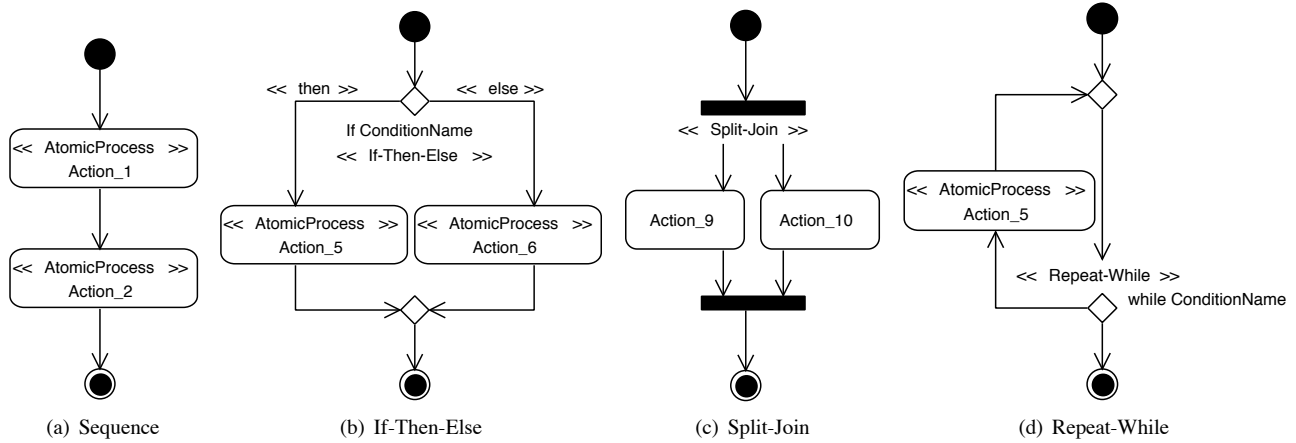


Figure 4. UML Activity Diagram Patterns of OWL-S Constructs

Tag Definition	Tagged Value Syntax (pseudo-BNF)
hasPrecondition	<precondition-name>:<ocl-expression> <precondition-name>
hadDataFrom	<input-name>,<var-name>,<process-name> <input-name>,<data-value>
hasLocal	<local-name>,<parameter-type>
hasResult	<result-name>
inCondition	<result-name>,<condition-name>:<ocl-expression> <result-name>,<condition-name>
hasResultVar	<result-name>,<result-var-name>,<parameter-type>
withOutput	<result-name>,<output-name>,<var-name>,<process-name> <output-name>,<data-value>
producedOutput	<output-name>,<var-name>,<process-name> <output-name>,<data-value>
hasEffect	<result-name>,<effect-name>:<ocl-expression> <result-name>,<effect-name>

where <data-value> := 'string' | number

Table 2. OCL Tagged Value Syntax

The SWRL language specification [6] was used as a guideline to develop transformation rules (encoded in XSLT) that transform OCL expressions into SWRL. The SWRL language uses a conjunction of atoms to represent expressions. There are four primary types of atoms which are supported: class atoms, individual property atoms, data-valued property atoms, and built-in atoms. An individual property atom holds true if an OWL individual is related to another OWL individual via a property. A class atom holds true if an OWL individual is a member of the specified OWL class. A data-valued property atom holds true if the property of an individual has a particular data value. The data value can be one of the following: an OCL number, an OCL string or the name of a variable. Built-in atoms are used to provide access to built-in functionality such as math functions, string functions, and comparison operators. Our transformation system provides support for the following comparison operators: *equal*, *notEqual*, *greaterThan*, *lessThan*, *greaterThanOrEqual*, and *lessThanOrEqual*. The OCL syntax for the various types of atoms used in a SWRL expression is shown in Table 3. The first column contains the some of the SWRL atom types supported by the framework. The OCL syntax for each SWRL atom type is seen in the second column. The third column contains a concrete

example of the OCL syntax for each atom type.

In our OCL-based representation, multiple atoms are conjuncted together using the *and* keyword to form an atom list. For example, consider the expression *condition1 and condition2 and condition3* where *condition1*, *condition2*, and *condition3* can be any of the atom types listed in Table 3. The XSLT transformation tokenizes the OCL expression using *and* as the delimiter and generates the corresponding SWRL in RDF/XML concrete syntax.

3.5. Execution

Currently we are using the OWL-S API [12] to execute services. Once the ungrounded specification has been generated from the specification tool, a grounding is created using our grounding tool [14]. The grounded specification is then loaded into our execution tool which uses the OWL-S API to execute the service. The GUI of the execution tool, monitors the execution and displays real-time feedback to the user.

4. Example

In this section, we use the FindCheaperBook example [15] to demonstrate the composite process specification approach for OWL-S. FindCheaperBook is a composite service for finding the cheaper of two books returned by two different atomic services. The service is composed of BookFinderProcess, an atomic process that accepts a book name as input and returns information about that book. AmazonPriceProcess is an atomic process that accepts book information as input and returns a price. BNPriceProcess is an atomic process that accepts book information as input and returns a price. ComparePricesProcess is a composite process that compares the two prices and returns the lower price and the name of the bookstore that it came from (e.g. Amazon [16] or Barnes & Noble [17]). The OWL-S *Split-Join* and *If-Then-Else* constructs are used in this example. The UML model used to represent the FindCheaperBook example contains a class diagram shown in Fig-

SWRL Atom Type	OCL Syntax	Example
Class Atom	instance.ocllsType(class)	aCustomer.ocllsType(Person)
Individual Property Atom	instance.property = instance	aPerson.hasFather = aFather
Datavalued Property Atom	instance.property = 'string'	aPerson.hasPhoneNumber = '555-1234'
BuiltIn Atom (lessThan)	var1 < var2	CP_BNPriceAmount < CP_AmazonPriceAmount

Table 3. SWRL Atoms and OCL Syntax Mappings

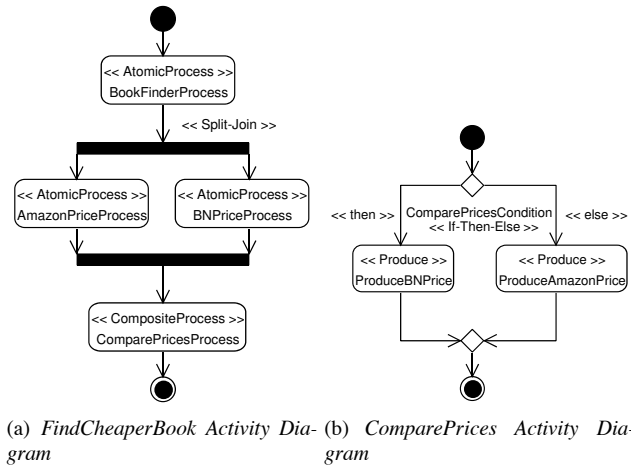


Figure 6. UML activity diagrams for example

ure 3 and two activity diagrams shown in Figure 6. The class diagram contains four classes: *FindCheaperBookService*, *FindCheaperBookProfile*, *FindCheaperBookProcess*, and *FindCheaperBookGrounding*. The *FindCheaperBookProcess* class contains two operations that are stereotyped using the `<< CompositeProcess >>` stereotype. The *FindCheaperBookProcess* operation is used to represent the entire process. The *ComparePricesProcess* represents the composite process which compares book prices. The process class also contains three operations that are stereotyped using the `<< AtomicProcess >>` stereotype. These operations represent the atomic processes used in the composition to find book information and prices. Finally, the process class contains two operations which are stereotyped using the `<< Produce >>` stereotype. This stereotype is used to mark an operation as being a pseudo-step in the composition that is used to bind output parameters. In this example, outputs of the comparison process are conditionally bound to the main process using *Produce* constructs. The UML Packages seen in the Figure 3 are used to refer to the *Price* and *Book* classes which are located in external ontologies.

The example contains three conditions: two preconditions (*AmazonPriceAmountCondition* and *BNPriceAmountCondition*) and the condition used in an *If-Then-Else* construct (*ComparePricesCondition*). All three conditions are used in the *ComparePricesProcess* composite process model. The OCL expressions used to represent these conditions are shown in Table 4. The first column of this table contains the name of the OCL condition from the example. The second column contains the condition in OCL syn-

tax. The OCL conditions were added to the UML model as tagged values using the syntax described in Section 3. The OCL was then parsed and transformed into SWRL expressions which were inserted into the transformation output.

OCL Condition Name	OCL Condition
AmazonPriceAmountCondition	CP_AmazonPrice.amount = CP_AmazonPriceAmount
BNPriceAmountCondition	CP_BNPrice.amount = CP_BNPriceAmount
ComparePricesCondition	CP_BNPriceAmount < CP_AmazonPriceAmount

Table 4. OCL conditions used in example

The OWL-S API [12] was used to validate the generated specification. The OWL-S API is a Java API for creating, validating, and executing OWL-S specifications. The specification generated by our transformation system was loaded into the OWL-S validator, which validated that the OWL ontology was consistent and that the OWL-S specification was valid. In addition, we are able to execute the specification using the OWL-S API. For a broader view of the research, including additional examples, please see our previous work in [13].

5. Related Work

The Business Process Execution Language for Web Services (BPEL4WS) [3] is an XML-based language for specifying web service compositions. BPEL4WS does not have the capabilities of languages such as OWL-S in that it does not support reasoning or precondition/postcondition evaluation. Conditions within control constructs in BPEL4WS are evaluated simply based on the structure and value of XML content defined for the input and output messages of a service. BPEL4WS also does not support the evaluation of pre and post conditions.

The OWL-S editor is a plug-in for the Protégé development environment that allows users to develop instances of the OWL-S ontology for services in a graphical manner [18]. The OWL-S Editor supports composition through the use of a visual editor. The editor is centered on the OWL-S language and therefore, all compositions are built using OWL-S control constructs. Our tools work with standard UML modeling tools. Services can thus be developed using the standard visual notation provided by UML.

Grønmo et al. have developed an approach for performing transformations from OWL-S specifications to UML activity diagrams [19]. In their work, they demonstrate how the specifications of the ExpressCongoBuy and Full-

CongoBuy examples from DAML [4] can be manually converted to UML activity diagrams utilizing input/ output pins and object flow features to model inputs, outputs, and data binding specifications. As such, their work can be considered more of a reverse engineering technique. In our work, we are using UML class diagrams for the general structure of the web service including the process inputs and outputs. We use tagged values on UML class diagrams for data binding and UML activity diagrams to represent control flow for composite processes. Taking these diagrams, we generate the OWL-S specification.

6. Conclusions and Future Work

Semantic Web Services can potentially change the way software is both developed and used. In order to realize that great promise, the software development community must embrace the technology. The barriers to adoption must be bridged in a manner that leverages the capabilities of developers. We have been developing an MDA-based approach for facilitating such adoption by using UML to specify web-based applications and transformations to achieve synthesis of semantic web services. Our previous investigations into the specification of atomic processes as well as OWL-S groundings [14] coupled with the work described in this paper provides movement towards a usable framework for semantic web services. Our current investigations involve developing the framework in such a way that it more adequately hides the details of semantic web service technologies and allows the developer to focus on creating models of semantic web service compositions. In addition, to supporting the specification task, we are actively pursuing the development of an end-to-end execution environment that will allow developers to specify, generate, and execute web-based applications based on the use of semantic web services.

References

- [1] Tim Berners-Lee, James Hendler, and Ora Lassila. The semantic web. *Scientific American*, 284:35–43, May 2001.
- [2] 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.
- [3] Francisco Curbera, Yaron Goland, Johannes Klein, Frank Leymann, Dieter Roller, Satish Thatte, and Sanjiva Weerawarana. Business process execution language for web services. [Online] Available <http://www-128.ibm.com/developerworks/library/specification/ws-bpel/>, July 2002.
- [4] 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.
- [5] The Object Management Group. Ocl 2.0 specification. [Online] Available <http://www.omg.org/docs/ptc/05-06-06.pdf>, June 2005.
- [6] Ian Horrocks, Peter F. Patel-Schneider, Harold Boley, Said Tabet, Benjamin Grosz, and Mike Dean. Swrl: A semantic web rule language combining owl and ruleml. [Online] Available <http://www.w3.org/Submission/SWRL/>, May 2004.
- [7] The Object Management Group. Ontology definition meta-model. [Online] Available <http://www.omg.org/docs/ad/05-08-01.pdf>, August 2005.
- [8] Kilian Kiko and Colin Atkinson. Integrating Enterprise Architecture Representation Languages. In *Proceedings of the Workshop on Vocabularies, Ontologies, and Rules for The Enterprise*, pages 41–50, September 2005.
- [9] The Object Management Group. The unified modeling language. [Online] Available <http://www.omg.org/cgi-bin/doc?formal/05-07-04>, July 2005.
- [10] Gerald C. Gannod and Sushant Bhatia. Facilitating automated search for web services. In *ICWS*, pages 761–764, 2004.
- [11] Katia P. Sycara, Massimo Paolucci, Anupriya Ankolekar, and Naveen Srinivasan. Automated discovery, interaction and composition of semantic web services. *J. Web Sem.*, 1(1):27–46, 2003.
- [12] The MINDSWAP Group. Owl-s api. [Online] Available <http://www.mindswap.org/2004/owl-s/api/>.
- [13] Gerald C. Gannod, John T.E. Timm, and Raynette J. Brodie. Facilitating the Specification of Semantic Web Services Using Model-Driven Development. *Journal of Web Services Research*, 3(3):61–81, 2006.
- [14] Gerald C. Gannod, Raynette J. Brodie, and John T.E. Timm. An interactive approach for specifying owl-s groundings. In *Proceedings of the 9th IEEE EDOC Enterprise Computing Conference (EDOC'05)*, pages 251–260, September 2005.
- [15] The MINDSWAP Group. Mindswap. [Online] Available <http://www.mindswap.org>.
- [16] Amazon.com, Inc. The amazon web service. [Online] Available <http://webservices.amazon.com/AWSECommerceService/AWSECommerceService.wsdl>.
- [17] Barnes & Noble, Inc. The barnes & noble web service. [Online] Available <http://www.abundanttech.com/webservices/bnprice/bnprice.wsdl>.
- [18] 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 2nd European Semantic Web Conference*, May 2005.
- [19] Roy Grønmo, Michael C. Jaeger, and Hjørdis Hoff. Transformations between uml and owl-s. In *Proceedings of the European Conference on Model Driven Architecture - Foundations and Applications (ECMDA-FA)*, November 2005.