Journal of Web Engineering, Vol. 9, No. 3 (2010) 243–265
© Rinton Press

TOWARD SEMANTIC WEB SERVICES AS MVC APPLICATIONS: FROM OWL-S VIA UML

CÁSSIO V. S. PRAZERES, MARIA DA GRAÇA PIMENTEL
Departamento de Ciências da Computação, Universidade de São Paulo
Av. Trabalhador São-carlense, 400 - São Carlos, São Paulo, Brazil
prazeres@icmc.usp.br, mgp@icmc.usp.br

ETHAN V. MUNSON
Dept. of EECS, University of Wisconsin - Milwaukee
Milwaukee, WI, 53201, USA
munson@uwm.edu

CESAR A. C. TEIXEIRA
Departamento de Ciências da Computação, Universidade Federal de São Carlos
Rod. Washington Luis, Km 235 - São Carlos, São Paulo, Brazil
cesar@dc.ufscar.br

Received June 21, 2009
Revised January 6, 2010

OWL-S is an application of OWL, the Web Ontology Language, that describes the semantics of Web Services so that their discovery, selection, invocation and composition can be automated. The research literature reports the use of UML diagrams for the automatic generation of Semantic Web Service descriptions in OWL-S. This paper demonstrates a higher level of automation by generating complete complete Web applications from OWL-S descriptions that have themselves been generated from UML.

Previously, we proposed an approach for processing OWL-S descriptions in order to produce MVC-based skeletons for Web applications. The OWL-S ontology undergoes a series of transformations in order to generate a Model-View-Controller application implemented by a combination of JavaBeans, JSP, and Servlets code, respectively. In this paper, we show in detail the documents produced at each processing step. We highlight the connections between OWL-S specifications and executable code in the various Java dialects and show the Web interfaces that result from this process.

Keywords: Design, Implementation, UML, XMI, MVC, Model-View-Controller, OWL, OWL-S, Semantic Web Services

1 Introduction

OWL, the Web Ontology Language, has been developed by the World Wide Web Consortium as a semantic markup language for publishing and sharing ontologies on the World Wide Web [11]. It has been applied in several areas, for instance for defining ontologies for Legal Case-based Reasoning (LCBR) systems [34], and to describe a chemical knowledge representation enabling precise molecular descriptions upon which reasoning can be applied in a logically valid manner [22].

Semantic Web Services offer the opportunity to enhance the automation of Web Services discovery, selection, invocation and composition. The Ontology Web Language for Services (OWL-S) is designed to allow rich semantic specifications to be associated with Web Services by formalizing three essential types of knowledge about a service: "what the service does", "how the service works" and "how to access the service" [27]. However, the deployment of semantic-rich description languages such as OWL-S is not trivial [32], since developers must learn how to formally describe service semantics using three classes: the *Service Profile* class describes "what the service does"; the *Service Model* class describes "how the service works"; and the *Service Grounding* class describes "how to access the service" [27].

Given the substantial learning curve for OWL in general, and for OWL-S in particular, some researchers have proposed that Semantic Web Services descriptions be generated automatically from UML diagrams (e.g. [20]). If it were also possible to generate Web applications directly from OWL-S descriptions, then it might be possible to generate complete applications directly from the widely understood UML representation.

From a Web engineering perspective, results reported by García et al. [13] and Distante et al. [12] are representative of efforts supporting the separation of concerns demanded in the design of applications which exploit the Model-View-Controller (MVC) architectural pattern[a] in Smalltalk [23]. The use of MVC is not recent, in fact it has been widely used in the construction of Web applications in the last decade (e.g. [8, 16, 21]).

In previous work we have proposed an approach for producing MVC-based skeletons for Web applications from OWL-S descriptions [31]. Our approach requires that the Semantic Web Service was originally modeled with UML diagrams and is based on the following:

- the UML diagrams, represented as XMI (XML Metadata Interchange)[b], are used to generate OWL-S descriptions by applying XSLT transformations;
- the OWL-S *Service Model* class is translated, via the OWL-S API, into Servlets that correspond to the mapping of Web Service composite processes to an MVC *Controller*;
- the OWL-S *Service Profile* class is translated, again via the OWL-S API into XML Schemas corresponding to the Web Service inputs and outputs. These Schemas are used to generate: (a) JavaBeans classes that correspond to an MVC *Model*; and (b) JSP code generating HTML that corresponds to an MVC *View*.

We illustrate our approach by showing, in detail, the generation of the JavaBeans, Servlets and JSP code for a fictitious airline site that is a standard example in the OWL-S literature. In this paper, we extend our previous work [31] by discussing an extensive set of documents produced as part of the generation of the MVC-based application.

This paper is organized as follows. In Section 2 we review the main concepts of the Ontology Web Language for Services (OWL-S). In Section 3 we outline related work that presents approaches for generating Semantic Web Services Descriptions from UML diagrams. In Section 4 we review our preliminary efforts toward exploiting XSL Transformations *from* XML Schemas *to* Web-based user interfaces. In Section 5 we detail our approach to the generation of MVC Web applications from OWL-S descriptions. In Section 6 we situate our

[a] Originally deployed http://st-www.cs.uiuc.edu/users/smarch/st-docs/mvc.html
[b] http://www.omg.org/technology/documents/formal/xmi.htm

proposal with respect to related work. We present our final remarks and discuss future work in Section 7.

2 Semantic Web Services and OWL-S

In this section we present OWL-S concepts that are relevant to our approach. Figure 1 (adapted from [27]) shows the upper-level OWL-S ontology for services.

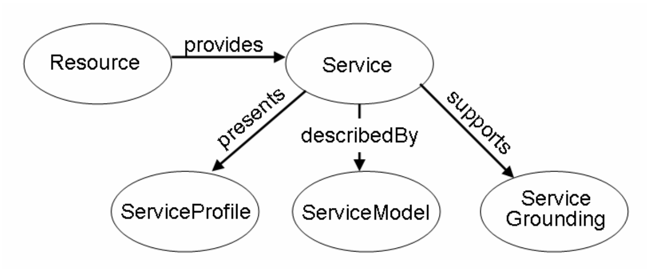


Figure 1: OWL-S Ontology for Services (adapted from [27]).

The OWL-S upper ontology for services is designed to provide three essential types of knowledge about a service, each represented by a class: the *ServiceProfile*, the *ServiceModel* and the *ServiceGrounding* classes [27]. In OWL-S, the knowledge about a service must be formally specified by the developer as follows: (a) the class *Service Profile* must describe "what the service does" by formalizing the inputs, outputs, preconditions and effects of the service; (b) the class *Service Model* describes "how the service works" by formalizing the composition of the service based on processes and control constructs such as sequences; (c) the class *Service Grounding* describes "how to access the service" by specifying information such as communication protocols (e.g. SOAP) and message formats (e.g. WSDL) [27].

The *ServiceProfile* class is the superclass of every type of high-level description of the service and is used by providers to publish services and by requesters to discover services. The OWL-S Profile focuses on two aspects of the service functionality: (a) *inputs* and *outputs* represent information transformation; (b) *preconditions* and *effects* represent state changes produced by the execution of the service. Together, these four elements of the ServiceProfile are referred to as the IOPE.

To understand how a service operates, the service is better viewed as a process. So, OWL-S defines a subclass of *ServiceModel*, the *ProcessModel*. A process in OWL-S has two functions. One is to produce a data transformation **from** a set of inputs **to** a set of outputs. Second, to produce a transition in the world **from** one state **to** another. This transition is described by means of the preconditions and the effects of the process. Clearly, there is a close relationship between a service's process model and its profile.

The *Process* class has three types of process sub-classes: Atomic, Simple, and Composite. Atomic processes are directly invocable (by passing them the appropriate messages), have no subprocesses, and from the perspective of the service requester, execute in a single step. An Atomic process connected to real operations by a Service Grounding. Simple processes are not invocable and are not associated with a grounding, but like atomic processes, they are conceived of as having single-step executions. They are container processes that represent an abstraction with only one sub-process. Composite processes are decomposable into other (non-composite or composite) processes. Their decomposition can be specified by using control constructs such as *Sequence* and *If-Then-Else* [27]. Like Simple processes, Composite processes are not grounded and are not invocable.

To generate Web applications in a MVC architecture, we map the constructors of OWL-S (*e.g.* inputs, outputs, process and control constructs) to Java code.

3 From UML to OWL-S

Grønmo et al. [15] propose a UML profile for semantic Web service composition, with the goal of supporting transformations both ways between UML and OWL-S. Their work is representative of efforts that claim that UML can be used as an integration platform and to support developers in the description and composition of Web services.

Research exploring the use of UML in the development of Semantic Web Service can be divided into two types: (a) research that uses UML to model Semantic Web Services applications [1, 5, 6], and (b) research that uses UML diagrams to automatically generate Semantic Web Services descriptions [20, 24, 32, 36]. It is the second type that is particularly relevant to the research presented in this paper.

Kim and Lee [20] proposed that UML class diagrams could represent a domain ontology, and that a UML sequence or activity diagrams could represent the behavior of a business process. They introduced a method to generate OWL-S descriptions by applying XSLT transformations to UML diagrams represented by XMI.

Yang and Chung [36] presented a methodology for the automatic generation of OWL-S descriptions from UML diagrams. They proposed that information about atomic services and their properties could be extracted from UML class diagrams as IOPE classes[c] and that information about composition of services could be extracted from UML statechart diagrams.

Lee et al. [24] presented a framework to support the evolution of Web applications based on Semantic Web Services that includes a method to derive service descriptions from UML use-case diagrams.

Timm and Gannod [32] presented an automated software tool that uses model-driven architecture techniques to generate an OWL-S description of a Web Service from a UML diagram.

In the research presented here, we generate OWL-S specifications from UML diagrams using techniques similar to those of Kim and Lee [20]. These OWL-S specifications are then used to generate MVC-structured Web applications.

4 From XML Schemas to Web-based User Interfaces via XSLT

In order to generate the MVC-Views for Web applications in the MVC architecture (as shown in detail in Section 5.4), our proposed approach builds upon our previous work in which we used XSLT style sheet transformations for the design of Web-based interfaces to WebLabs [29]. WebLabs are laboratories that allow resources to be remotely accessed by means of experiments controlled via a computer network in general, and making use of Web based interfaces in particular.

[c] OWL-S classes for Web Service inputs, outputs, preconditions and effects.

Since a wide variety of WebLabs and experiments can be built, we have designed an approach that facilitates the extension and customization of Web-based applications according to the requirements of WebLabs. This has been achieved by extensive use of XML Schemas to structure both the *a priori* information applicable to the entire WebLabs application domain and the specific, on-the-fly information that accommodates the unique characteristics of each lab.

Support for the on-the-fly document-based specification is achieved by XSLT transformations that allow the generation of the required presentation documents. The on-the-fly processing of documents is implemented within document-processing flows involving XML Schema specifications, XSLT transformations and form-based presentation documents, as shown in Figure 2.

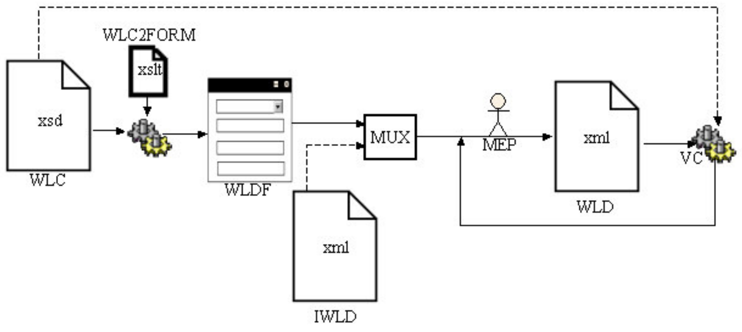


Figure 2: Document-based flow for WebLab registration (adapted from [29])

.

In Figure 2, the flow starts with the processing of an XML Schema containing information common to most WebLabs; the result is the generation of a form to be completed by the WebLab administrator with the information corresponding to his lab. Once the form is completed, the corresponding lab specification is validated in order to guarantee conformance with data types and structure. More specifically, in Figure 2:

- the XML Schema is indicated by WLC: Web Lab Concept;
- the transformation document is indicated by WLC2FORM: WLC to Form;
- the form presented to the WebLab Administrator is indicated by WLDF: WebLab Description Form;
- the completion of the form by the user is indicated by MEP: Manual Editing Process;
- the instance document produced as a result of the edition is indicated by WLD: WebLab Description;
- the validation of the information provided by the user is indicated by VC: validation check;

- the provision of an alternative instance document, possibly from reuse, is indicated by IWLD: Imported WLD;
- the selection between the IWLD document or the information provided manually (via the manual filling of the forms MEP) is indicated with the MUX: multiplex selector.

The XML Schema presented below in Document 1 illustrates, as an example, the type of information that must be specified for each lab (WLC: Web Lab Concept in Figure 2). The processing of this document with an appropriate XSLT transformation (WLC2FORM in Figure 2) generates the form presented in Figure 3 (WDLF in Figure 2).

```
<!--        Document 1        -->
<!-- A segment of the WLC XML-Schema -->

<?xml version="1.0" encoding="ISO-8859-1" ?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
 <xsd:simpleType name="nameType">
  <xsd:restriction base="xsd:string">
   <xsd:maxLength value="20" />
  </xsd:restriction>
 </xsd:simpleType>
 <xsd:simpleType name="descriptionType">
  <xsd:restriction base="xsd:string">
   <xsd:maxLength value="400" />
  </xsd:restriction>
 </xsd:simpleType>
 <xsd:complexType name="webladType">
  <xsd:sequence>
   <xsd:element name="name" type="nameType" minOccurs="1" maxOccurs="1"/>
   <xsd:element name="url" type="urlType" minOccurs="1" maxOccurs="1"/>
   <xsd:element name="description" type="descriptionType" maxOccurs="1"/>
  </xsd:sequence>
  <xsd:attribute name="WebLabId" type="xsd:ID" use="required" />
 </xsd:complexType>
 <xsd:element name="WebLab" type="webladType" />
</xsd:schema>
```

This section has shown how we used XSLT transformations for the design of Web-based interfaces to WebLabs. In the following sections, we build upon this transformation approach to generate JSP code implementing the MVC-View component of Web applications by processing OWL-S specifications.

5 From OWL-S to an MVC-based Web Application

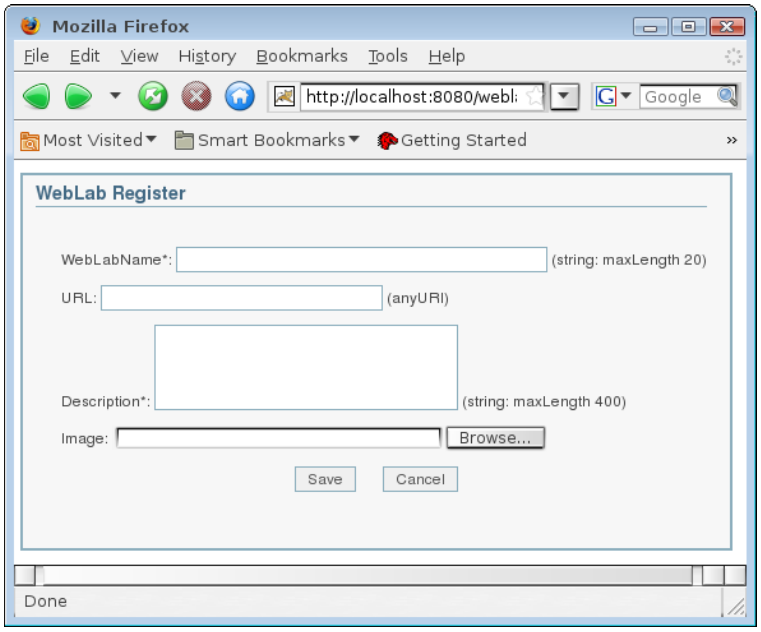
In a MVC architecture, the core business model functionality is separated from the presentation and control logic. This architecture allows the same enterprise data model to have multiple views.[d] The goal is to decouple the graphical interface from the navigation and behavior of the application. This decoupling promotes easier maintenance and greater reuse.

There are several variations of the MVC architecture in the literature: we use the approach that does not permit direct interaction between View and Model.

5.1 OWLS2MVC processing overview

Figure 4 presents the workflow for generating MVC-based Web applications from OWL-S descriptions. As shown in Figure 4, the generation process includes four steps including *OWL-S Generation*, *Ontology Data Extraction*, *Schema Data Extraction* and *XSLT Transformation.*

[d]http://java.sun.com/blueprints/patterns/MVC-detailed.html



WebLab Register

WebLabName*:		(string: maxLength 20)
URL:		(anyURI)
Description*:		(string: maxLength 400)
Image:		

Figure 3: Sample WebLab Description Form (WLDF).

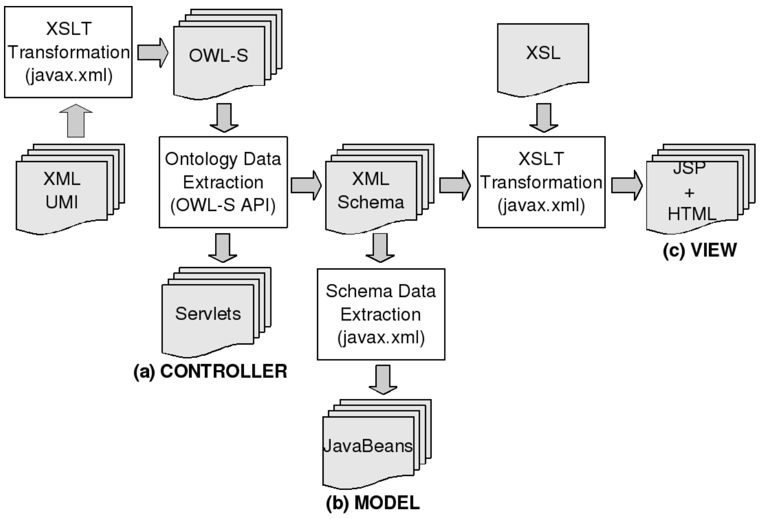

XSLT Transformation (javax.xml)
OWL-S
XSL
XML UMI
Ontology Data Extraction (OWL-S API)
XML Schema
XSLT Transformation (javax.xml)
JSP + HTML
(c) VIEW
Servlets
(a) CONTROLLER
Schema Data Extraction (javax.xml)
JavaBeans
(b) MODEL

Figure 4: OWLS2MVC generation overview

In the first step of the process, we use the approach proposed by Kim and Lee [20] to automatically generate OWL-S descriptions from UML diagrams by applying XSLT transformations to XMI documents. However, the OWLS2MVC processing can also be initiated directly from OWL-S descriptions, which corresponds to the second step in our approach, without needing to use Kim and Lee's approach.

In the process's second step, data from OWL-S descriptions is extracted using the OWL-S API[e]in order to generate two documents: the Servlets that implements the MVC Controller (Figure 4(a)), and XML Schemas that correspond to the inputs and outputs of a service in OWL-S.

In the third step, these XML Schemas are transformed into JavaBeans that correspond to the Model component of the MVC architecture (Figure 4(b))

The fourth step involves the generation of JSP code that renders HTML documents that map to the View components of the MVC architecture (Figure 4(c)).

Table 1 shows the mapping relationships between OWL-S and Java code. All composite processes in OWL-S generate a Servlet class for the Controller components in our MVC architecture: the Servlet Controller as explained in Section 5.2. Atomic processes that compose the composite process are mapped into methods inside the Servlet Controller. The workflow of each Servlet or each method is generated by the control constructs (such as If-Then-Else and Repeat-While detailed in Table 1) of OWL-S.

Table 1: OWL-S to Java Mapping

Type	**OWL-S**	**Java Code**
Process	composite	Servlet Class
	atomic	Method
Control Construct	Sequence	sequence of code
	If-Then-Else	if-then-else
	Choice	switch
	Repeat-While	while
	Rapeat-Until	do-while
Parameter	Input	MVC Model
	Output	MVC Model
	parameterType	type of attributes

Table 1 also shows that the Models in our MVC architecture are generated from the service inputs and outputs. These inputs and outputs become the arguments to a Java method in the Servlet Controller. Their generation is explained in Section 5.3.

Sections 5.2, 5.3 and 5.4 describe how each file of the MVC architecture is generated using the classic example of OWL-S: Bravo Air[f]— a fictitious airline site.

5.2 Controller

The second step of our OWLS2MVC flow generates Java files that are Servlets classes. These Servlets are the Controller components in our MVC architecture. The Controller is generated by extracting information about the composite and atomic processes from the OWL-S specification.

Document 2 presents the composite process *BravoAir_Process* in OWL-S. This process is composed of a sequence of two atomic process (lines 8 and 9 in Document 2) and a composite process (line 10 in Document 2). The composite processes are mapped into Servlet Controllers

[e]http://projects.semwebcentral.org/projects/owl-s-api/

[f]http://www.daml.org/services/owl-s/1.0/examples.html

and the atomic processes are mapped into Java methods in the Servlets as detailed in sections 5.2.1 and 5.2.2.

In other words, the *BravoAir* service is defined by means of three processes as follows: the first, *GetDesiredFlightDetails*, is responsible for searching flights based on the information provided by its users, the second, *SelectAvailableFlight*, selects one flight among those available; and the third service, *BookFlight*, is responsible for the scheduling of the selected flight.

```
0  <!--        Document 2        -->
1  <!-- The main composite process in the BravoAir Service -->
2
3  <process:CompositeProcess rdf:ID="BravoAir_Process">
4   <rdfs:label> This is the top level process for BravoAir </rdfs:label>
5   <process:composedOf>
6    <process:Sequence>
7     <process:components rdf:parseType="Collection">
8      <process:AtomicProcess rdf:about="#GetDesiredFlightDetails"/>
9      <process:AtomicProcess rdf:about="#SelectAvailableFlight"/>
10     <process:CompositeProcess rdf:about="#BookFlight"/>
11    </process:components>
12   </process:Sequence>
13  </process:composedOf>
14 </process:CompositeProcess>
```

5.2.1 BravoAir Servlet

The composite process *BravoAir_Process* presented in Document 2 is the main process of the Bravo Air Service. In our OWLS2MVC approach, the composite process *BravoAir_Process* is mapped into a Servlet Controller as follows.

Document 3 presents the definition of the atomic process *GetDesiredFlightDetails*. This process is one of the three processes that form the composite process *BravoAir_Process*. Lines 4 to 8 of Document 3 describe the inputs for the atomic process. This atomic process will be mapped into one Java method in the Servlet Controller as shown in Document 5 (line 23).

```
0  <!--        Document 3        -->
1  <!-- The atomic process GetDesiredFlightDetails -->
2
3  <process:AtomicProcess rdf:ID="GetDesiredFlightDetails">
4   <process:hasInput rdf:resource="#DepartureAirport_In"/>
5   <process:hasInput rdf:resource="#ArrivalAirport_In"/>
6   <process:hasInput rdf:resource="#OutboundDate_In"/>
7   <process:hasInput rdf:resource="#InboundDate_In"/>
8   <process:hasInput rdf:resource="#RoundTrip_In"/>
9  </process:AtomicProcess>
```

Document 4 presents the definition of the atomic process *SelectAvailableFlight*. This process is one of the three process that form the composite process *BravoAir_Process*. Lines 4 and 5 of Document 4 describe the inputs for the atomic process. This atomic process will be mapped into one Java method in the Servlet Controller as shown in Document 5 (line 26).

```
0  <!--        Document 4        -->
1  <!-- The atomic process SelectAvailableFlight -->
2
3  <process:AtomicProcess rdf:ID="SelectAvailableFlight">
4   <process:hasInput rdf:resource="#PreferredFlightItinerary_In"/>
5   <process:hasOutput rdf:resource="#AvailableFlightItineraryList_Out"/>
6  </process:AtomicProcess>
```

As indicated in Document 5, a Servlet extends the *HttpServlet* class and implements the *Servlet* interface. The document also shows that the Servlet implements the methods *doGet* and/or *doPost*. These methods can be invoked when a HTTP request is made by the client browser.

Document 2 describes a composite process as a sequence of three process (two atomics and one composite). This sequence is mapped into a sequence of two Java methods (the atomic process) and one Servlet (the composite process), and is located inside the methods *doGet* and *doPost* of the Servlet Controller (lines 11 to 13 and 18 to 20 in Document 5). Lines 23 to 27 in Document 5 correspond to the atomic processes in the Servlet Controller: *getDesiredFlightDetails* and *getDesiredFlightDetails* (Documents 3 and 4).

```
0  <!--        Document 5        -->
1  <!-- Servlet Controller for the composite process: BravoAir_Process -->
2
3  public class BravoAirServletController extends HttpServlet implements Servlet {
4
5   public BravoAirServletController() {
6    super();
7   }
8
9   protected void doGet(HttpServletRequest request,
    HttpServletResponse response) throws ServletException, IOException {
10   // process Sequence of OWL-S
11   getDesiredFlightDetails(request, response);
12   selectAvailableFlight(request, response);
13   response.sendRedirect (request.getContextPath() + "/BookFlight");
14  }
15
16  protected void doPost(HttpServletRequest request,
    HttpServletResponse response) throws ServletException, IOException {
17   // process Sequence of OWL-S
18   getDesiredFlightDetails(request, response);
19   selectAvailableFlight(request, response);
20   response.sendRedirect (request.getContextPath() + "/BookFlight");
21  }
22
23  private void getDesiredFlightDetails (HttpServletRequest request,
    HttpServletResponse response) throws ServletException {
24  }
25
26  private void selectAvailableFlight (HttpServletRequest request,
    HttpServletResponse response) throws ServletException {
27  }
28 }
```

As illustrated in Document 5 (lines 13 and 20), the workflow of the Servlet Controller is redirected to another Servlet: *BookFlight*. This Servlet corresponds to the composite process described in Document 2 (line 10). The *BookFlight* Servlet should be mapped into another Servlet Controller as described in Section 5.2.2.

5.2.2 Book Flight Servlet

The *BookFlight* process presented in Document 2 is a sub-process of the composite *BravoAir_Process*. In OWLS2MVC, the *BookFlight* process is mapped into a Servlet Controller as follows.

Document 6 presents the *BookFlight* composite process in OWL-S. *BookFlight* is composed of a sequence of two atomic processes (lines 7 and 31 in Document 6). The *ConfirmReservation* process, referred to on line 31, only executes if (constructor *If-Then-Else* on line 9) the

Login process was successfully executed. The *BookFlight* process is mapped into a Servlet Controller, while its sub-processes are mapped into Java method calls inside the Servlet, because they are atomic and are grounded to those Java methods. The type of each process is specified explicitly by the type of tag used to define each process (shown in Document 6, 7, and 8). The resulting Java code is shown in Document 9.

```
0  <!--        Document 6        -->
1  <!-- The composite process BookFlight -->
2
3  <process:CompositeProcess rdf:ID="BookFlight">
4   <process:composedOf>
5    <process:Sequence>
6     <process:components rdf:parseType="Collection">
7      <process:AtomicProcess rdf:about="#Login"/>
8
9      <process:If-Then-Else>
10      <process:ifCondition>
11       <expr:SWRL-Condition>
12        <rdfs:label>LoggedIn(AcctName)</rdfs:label>
13        <rdfs:comment>
14         This condition will be true if the previous PerformLogIn operation was succesful
15        </rdfs:comment>
16        <expr:expressionBody rdf:parseType="Literal">
17         <swrl:AtomList>
18          <rdf:first>
19           <swrl:ClassAtom>
20            <swrl:classPredicate rdf:resource="#LoggedIn"/>
21            <swrl:argument1 rdf:resource="#CompleteReservation_AcctName"/>
22           </swrl:ClassAtom>
23          </rdf:first>
24          <rdf:rest rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#nil"/>
25         </swrl:AtomList>
26        </expr:expressionBody>
27       </expr:SWRL-Condition>
28      </process:ifCondition>
29
30      <process:then>
31       <process:process rdf:resource="#ConfirmReservation"/>
32      </process:then>
33     </process:If-Then-Else>
34    </process:components>
35   </process:Sequence>
36  </process:composedOf>
37 </process:CompositeProcess>
```

Document 7 shows the atomic *Login* process that is part of the composite *BookFlight* process. *LogIn* requires two inputs as shown in lines 4 and 5 of Document 7. The process is mapped into one Java method call in the doPost and doGet methods of the Servlet corresponding to the *BookFlight* process as shown in Document 9.

```
0  <!--        Document 7        -->
1  <!-- The atomic process LogIn -->
2
3  <process:AtomicProcess rdf:ID="LogIn">
4   <process:hasInput rdf:resource="#AcctName_In"/>
5   <process:hasInput rdf:resource="#Password_In"/>
6  </process:AtomicProcess>
```

Document 8 shows the atomic *ConfirmReservation* process. *ConfirmReservation* needs two inputs and provides three outputs as shown in lines 4 and 8 of Document 8. Like the previous *Login* process, *ConfimReservation* is mapped into one Java method call in the doPost and doGet methods of the Servlet shown in Document 9.

```
<!--        Document 8        -->
<!-- The atomic process ConfirmReservation -->

<process:AtomicProcess rdf:ID="ConfirmReservation">
 <process:hasInput rdf:resource="#ReservationID_In"/>
 <process:hasInput rdf:resource="#Confirm_In"/>
 <process:hasOutput rdf:resource="#PreferredFlightItinerary_Out"/>
 <process:hasOutput rdf:resource="#AcctName_Out"/>
 <process:hasOutput rdf:resource="#ReservationID_Out"/>
</process:AtomicProcess>
```

Document 6 describes the composite process *BookFlight* as a sequence of two atomic process. This sequence is mapped into a sequence of two Java methods, and is located inside the methods *doGet* and *doPost* of the Servlet Controller (lines 11 to 15, and 21 to 25 in Document 9). Lines 28 to 39 in Document 9 correspond to the atomic processes in the Servlet Controller: *LogIn* and *ConfirmReservation* (Documents 7 and 8).

```
0  <!--        Document 9        -->
1  <!-- Servlet Controller for the composite process: BookFlight -->
2
3  public class BravoAirServletController extends HttpServlet implements Servlet {
4
5   public BookFlightServletController() {
6    super();
7   }
8
9   protected void doGet(HttpServletRequest request,
    HttpServletResponse response) throws ServletException, IOException {
10   // process Sequence of OWL-S
11   login(request, response);
12   // To Do: defines LoggedIn
13   if (LoggedIn) {
14        confirmReservation(request, response);
15   }
16
17  }
18
19  protected void doPost(HttpServletRequest request,
    HttpServletResponse response) throws ServletException, IOException {
20   // process Sequence of OWL-S
21   login(request, response);
22   // To Do: defines LoggedIn
23   if (LoggedIn) {
24        confirmReservation(request, response);
25   }
26  }
27
28  private void login (HttpServletRequest request,
    HttpServletResponse response) throws ServletException {
29  }
30
31  private void confirmReservation (HttpServletRequest request,
    HttpServletResponse response) throws ServletException {
32  }
33 }
```

According to the code shown in Document 6, the process *ConfirmReservation* is executed only if the atomic process *LogIn* is successfully concluded, in other words only if the user is logged in the service. Note that lines from 13 to 15 and from 23 to 25 in the Servlet presented in Document 9 contains the a conditional *if(LoggedIn)*, which corresponds to the constructor OWL-S *If-Then-Else* shown in line 9 of Document 6.

5.3 Model

In OWLS2MVC, the inputs and outputs of an OWL-S service process are always mapped to a Model in the MVC architecture. The Model is generated in the second and third steps of the processing. In these steps, the inputs and outputs are extracted from the OWL-S specification, an XML Schema is generated, and JavaBeans corresponding to the Model are created (see Figure 4). As far as the *BravoAir* service is concerned, only the atomic processes are associated with inputs and outputs, as is shown in the following sections.

5.3.1 Desired Flight Details Model

The atomic process *GetDesiredFlightDetails* is part of the composite process *BravoAir_Process* which is the main process of the *BravoAir* service. Document 2 shows the inputs for the OWL-S process *GetDesiredFlightDetails*. For example, the input *DepartureAirport_In* at lines 3 to 5 in Document 10 is the departure airport for the Bravo Air service.

```
0  <!--        Document 10        -->
1  <!-- Inputs for the process GetDesiredFlightDetails in OWL-S -->
2
3  <process:Input rdf:ID="DepartureAirport_In">
4   <process:parameterType rdf:resource="&concepts;#Airport"/>
5  </process:Input>
6
7  <process:Input rdf:ID="ArrivalAirport_In">
8   <process:parameterType rdf:resource="&concepts;#Airport"/>
9  </process:Input>
10
11 <process:Input rdf:ID="OutboundDate_In">
12  <process:parameterType rdf:resource="&concepts;#FlightDate"/>
13 </process:Input>
14
15 <process:Input rdf:ID="InboundDate_In">
16  <process:parameterType rdf:resource="&concepts;#FlightDate"/>
17 </process:Input>
18
19 <process:Input rdf:ID="RoundTrip_In">
20  <process:parameterType rdf:resource="&concepts;#RoundTrip"/>
21 </process:Input>
```

These inputs are defined by using XML Schemas as in Document 11. The *Airport* class (lines 3 to 5 in Document 11) defines the type of the input *DepartureAirport_In* (lines 3 to 5 in Document 10), and it has the datatype string defined by XML Schema datatypes[g]

```
0  <!--        Document 11        -->
1  <!-- Types definition for the Inputs -->
2
3  <owl:Class rdf:ID="Airport">
4   <rdfs:subClassOf rdf:resource="&xsd;#string"/>
5  </owl:Class>
6
7  <owl:Class rdf:ID="RoundTrip">
8   <rdfs:subClassOf rdf:resource="&xsd;#boolean"/>
9  </owl:Class>
10
11 <owl:Class rdf:ID="FlightDate">
12  <rdfs:subClassOf rdf:resource="&xsd;#date"/>
13 </owl:Class>
```

[g]http://www.w3.org/TR/xmlschema-2/

The XML Schemas generated in the third step are used to generate the Model as JavaBeans classes, as shown in Document 4. The lines 5 to 9 of Document 4 show some attributes of the JavaBeans class. These attributes are generated from the inputs of the OWL-S listed in Document 2. The XML Schema of Document 3 determines the type of each attribute in the JavaBean class. The lines 11 to 40 of Document 4 show all the get and set methods of the JavaBeans class.

It is important to observe that the *GetDesiredFlightDetails* process does not include outputs and, as a result, only code for one JavaBean for the specified inputs must be generated.

```
<!--         Document 12        -->
<!-- The JavaBeans for the Model of inputs for the process GetDesiredFlightDetails -->

public class DesiredFlightInBean {

 private String departureAirport_In;
 private String arrivalAirport_In;
 private Date outboundDate_In;
 private Date inboundDate_In;
 private boolean roundTrip_In;

 public String getArrivalAirport_In() {
  return arrivalAirport_In;
 }
 public void setArrivalAirport_In (String arrivalAirport_In) {
  this.arrivalAirport_In = arrivalAirport_In;
 }
 public String getDepartureAirport_In() {
  return departureAirport_In;
 }
 public void setDepartureAirport_In (String departureAirport_In) {
  this.departureAirport_In = departureAirport_In;
 }
 public Date getInboundDate_In() {
  return inboundDate_In;
 }
 public void setInboundDate_In(Date inboundDate_In) {
  this.inboundDate_In = inboundDate_In;
 }
 public Date getOutboundDate_In() {
  return outboundDate_In;
 }
 public void setOutboundDate_In(Date outboundDate_In) {
  this.outboundDate_In = outboundDate_In;
 }
 public boolean isRoundTrip_In() {
  return roundTrip_In;
 }
 public void setRoundTrip_In(boolean roundTrip_In) {
  this.roundTrip_In = roundTrip_In;
 }
}
```

5.3.2 Available Flight Model

Similar to previous cases, the atomic process *SelectAvailableFlight* is also part of the composite process *BravoAir_Process*. Document 13 presents the definition of the inputs and outputs corresponding to the *SelectAvailableFlight* process. The types of the inputs and outputs are defined by means of the constructor *parameterType* of OWL-S in lines 4 and 8 of Document 13. These types are used to define the corresponding types of the attributes in the JavaBeans model corresponding to the MVC.

```
0  <!--        Document 13        -->
1  <!-- Inputs and outputs for the process SelectAvailableFlight in OWL-S -->
2
3  <process:Input rdf:ID="PreferredFlightItinerary_In">
4   <process:parameterType
     rdf:resource="http://www.daml.org/services/owl-s/1.0/Concepts.owl#FlightItinerary"/>
5  </process:Input>
6
7  <process:UnConditionalOutput rdf:ID="AvailableFlightItineraryList_Out">
8   <process:parameterType
     rdf:resource="http://www.daml.org/services/owl-s/1.0/Concepts.owl#FlightItineraryList"/>
9  </process:UnConditionalOutput>
```

The input *PreferredFlightItinerary_In* of the process *SelectAvailableFlight* is mapped into one JavaBean corresponding to the (MVC) model for the inputs of the process. However, the process *SelectAvailableFlight* contains only one entry and, as a result, only this entry is declared in the JavaBean as shown in Document 14, line 5. Moreover, lines 7 to 12 in Document 14 present the get and set methods of the JavaBean corresponding to the input *PreferredFlight-Itinerary_In*.

```
0  <!--        Document 14        -->
1  <!-- The JavaBeans for the Model of inputs for the process SelectAvailableFlight -->
2
3  public class AvailableFlightInBean {
4
5   private String preferredFlightItinerary_In;
6
7   public String getPreferredFlightItinerary_In() {
8    return preferredFlightItinerary_In;
9   }
10  public void setPreferredFlightItinerary_In (String preferredFlightItinerary_In) {
11   this.preferredFlightItinerary_In = preferredFlightItinerary_In;
12  }
13 }
```

The input *AvailableFlightItineraryList_Out* of the *SelectAvailableFlight* process is mapped into one JavaBean corresponding to the model (Model in MVC) of the outputs of the process. However, the *SelectAvailableFlight* process contains only one output, which is declared in the JavaBeans presented in Document 15, line 5. Also, lines 7 to 12 of Document 15 present the get and set JavaBean methods corresponding to the output *AvailableFlightItineraryList_Out*.

```
0  <!--        Document 15        -->
1  <!-- The JavaBeans for the Model of outputs for the process SelectAvailableFlight -->
2
3  public class AvailableFlightOutBean {
4
5   private List availableFlightItineraryList_Out;
6
7   public List getAvailableFlightItineraryList_Out() {
8    return availableFlightItineraryList_Out;
9   }
10  public void setAvailableFlightItineraryList_Out (List availableFlightItineraryList_Out)
    {
11   this.availableFlightItineraryList_Out = availableFlightItineraryList_Out;
12  }
13 }
```

5.3.3 LogIn Model

The atomic process *LogIn* is part of the composite process *BookFlight* presented in Document 6. Document 16 details the definition of all inputs corresponding to the *LogIn* process. The types of the inputs are defined by means of the OWL-S *parameterType* constructor in lines from 4 to 8 in Document 16. These types are used to define the types of the attributes on the model (that is, the JavaBeans in MVC).

```
0  <!--        Document 16        -->
1  <!-- Inputs for the process LogIn in OWL-S -->
2
3  <process:Input rdf:ID="AcctName_In">
4   <process:parameterType
     rdf:resource="http://www.daml.org/services/owl-s/1.0/Concepts.owl#AcctName"/>
5  </process:Input>
6
7  <process:Input rdf:ID="Password_In">
8   <process:parameterType
     rdf:resource="http://www.daml.org/services/owl-s/1.0/Concepts.owl#Password"/>
9  </process:Input>
```

It is important to observe that the *LogIn* process does not have outputs and, as a result, only one JavaBean for the input must be generated. The two inputs of the *LogIn* process are mapped into one JavaBean corresponding to the (MVC) Model of the process inputs as stated in lines 5 and 6 of Document 17. Also in Document 17, lines from 8 t 20 show the get and set JavaBeans methods corresponding to the inputs of the *LogIn* process.

```
0  <!--        Document 17        -->
1  <!-- The JavaBeans for the Model of inputs for the process LogIn -->
2
3  public class LogInInBean {
4
5   private String acctName_In;
6   private String password_In;
7
8   public String getAcctName_In() {
9    return acctName_In;
10 }
11  public void setAcctName_In (String acctName_In) {
12   this.acctName_In = acctName_In;
13  }
14
15  public String getPassword_In() {
16   return password_In;
17  }
18  public void setPassword_In (String password_In) {
19   this.password_In = password_In;
20  }
21 }
```

5.3.4 Reservation Model

The atomic process *ConfirmReservation* is also part of the *BookFlight* composite process presented in Document 6. The definition of the inputs and outputs for the *ConfirmReservation* process is presented in Document 18. The types of the inputs and outputs is defined by means of the OWL-S *parameterType* constructor as shown in lines 4, 8, 12, 16 and 20 in Document 18. As in the previous examples, these types are used to define the attribute types in the model (the JavaBeans in the MVC pattern).

```
0  <!--       Document 18       -->
1  <!-- Inputs and outputs for the process ConfirmReservation in OWL-S -->
2
3  <process:Input rdf:ID="ReservationID_In">
4   <process:parameterType
     rdf:resource="http://www.daml.org/services/owl-s/1.0/Concepts.owl#ReservationNumber"/>
5   </process:Input>
6
7  <process:Input rdf:ID="Confirm_In">
8   <process:parameterType
     rdf:resource="http://www.daml.org/services/owl-s/1.0/Concepts.owl#Confirmation"/>
9  </process:Input>
10
11 <process:UnConditionalOutput rdf:ID="PreferredFlightItinerary_Out">
12  <process:parameterType
     rdf:resource="http://www.daml.org/services/owl-s/1.0/Concepts.owl#FlightItinerary"/>
13 </process:UnConditionalOutput>
14
15 <process:UnConditionalOutput rdf:ID="AcctName_Out">
16  <process:parameterType
     rdf:resource="http://www.daml.org/services/owl-s/1.0/Concepts.owl#AcctName"/>
17 </process:UnConditionalOutput>
18
19 <process:UnConditionalOutput rdf:ID="ReservationID_Out">
20  <process:parameterType
     rdf:resource="http://www.daml.org/services/owl-s/1.0/Concepts.owl#ReservationNumber"/>
21 </process:UnConditionalOutput>
```

The inputs of the process *ConfirmReservation* are mapped into one JavaBean corresponding to the (MVC) model of the inputs of the process, as defined in lines 5 and 6 in Document 19. Also, the lines from 8 to 20 in Document 19 present the get and set methods of the JavaBeans corresponding to the inputs of the process.

```
0  <!--       Document 19       -->
1  <!-- The JavaBeans for the Model of inputs for the process ConfirmReservation -->
2
3  public class ReservationInBean {
4
5   private int reservationID_In;
6   private boolean confirm_In;
7
8   public int getReservationID_In() {
9    return reservationID_In;
10  }
11  public void setReservationID_In (int reservationID_In) {
12   this.reservationID_In = reservationID_In;
13  }
14
15  public boolean getConfirm_In() {
16   return confirm_In;
17  }
18  public void setConfirm_In (boolean confirm_In) {
19   this.confirm_In = confirm_In;
20  }
21 }
```

The outputs of the process *ConfirmReservation* are also mapped into one JavaBean corresponding to the model (Model in MVC) of the outputs of the process as declared in lines 5, 6 and 7 in Document 20. Moreover, lines 9 to 28 in Document 20 define the get and set methods of the JavaBeans corresponding to the outputs of the process.

```
<!--       Document 20       -->
<!-- The JavaBeans for the Model of outputs for the process ConfirmReservation -->

public class ReservationOutBean {

 private String preferredFlightItinerary_Out;
 private String acctName_Out;
 private int reservationID_Out;

 public String getPreferredFlightItinerary_Out() {
  return preferredFlightItinerary_Out;
 }
 public void setPreferredFlightItinerary_Out (String preferredFlightItinerary_Out) {
  this.preferredFlightItinerary_Out = preferredFlightItinerary_Out;
 }

 public String getAcctName_Out() {
  return acctName_Out;
 }
 public void setAcctName_Out (String acctName_Out) {
  this.acctName_Out = acctName_Out;
 }

 public int getReservationID_Out() {
  return reservationID_Out;
 }
 public void setReservationID_Out (int reservationID_Out) {
  this.reservationID_Out = reservationID_Out;
 }
}
```

5.4 View

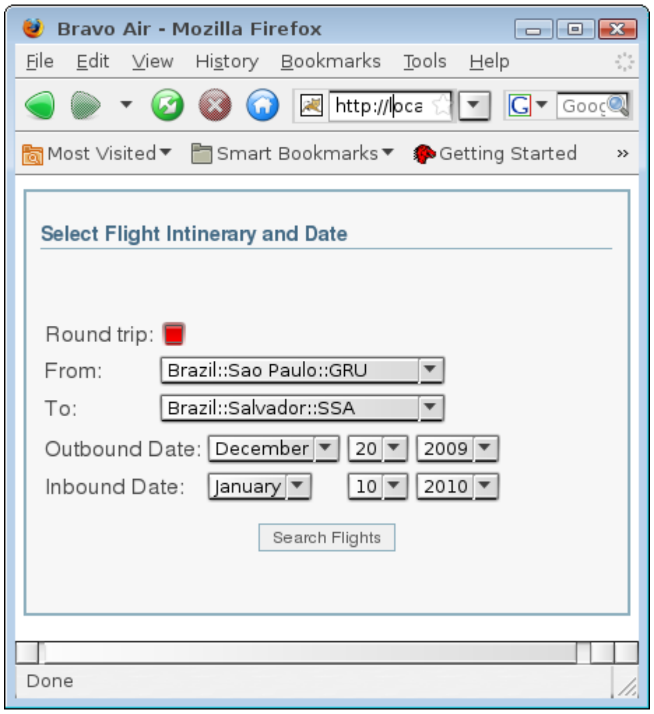
Figures 5 and 6 show examples of two possible automatically generated Views. Of course, there are many more possibilities. The Web developer can modify these Views or create other Views from scratch. The Views are generated in the fourth step of our workflow by applying XSLT transformations to the XML Schemas generated in the previous processing step. We use XSLT transformations applied to the XML Schemas generated for the Models in the third step. These transformations create JSP and HTML documents. For each Model at least one View is generated. All Views can be generated by using the approach presented in Section 4.

Figure 5 shows a form dialog that was generated by transformation from Document 10 and corresponds directly to the model that Document 10 represents. This form would be completed by a user who is searching for a flight and asks the user to provide information about the desired flight itinerary. Once submitted, the user's itinerary is automatically validated to guarantee conformance with datatypes and structure that will conform to the interfaces of the model generated in Document 12.

Figure 6 shows a second automatically generated form dialog through which a user could select outbound and return flights from a set of choices. This form dialog was generated from the Model presented in Document 13 using the same transformation approach and like the previous example dialog is automatically validated to guarantee it conforms with the model generated in Document 15.

6 Related Works

In the area of ontological engineering, several authors have proposed the use XSLT transformations from UML to generate ontologies, as in Cranefield's pioneering work generating RDF


Bravo Air - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://loca
Goog
Most Visited
Smart Bookmarks
Getting Started
Done

Select Flight Intinerary and Date

Round trip: 🟥
From: Brazil::Sao Paulo::GRU
To: Brazil::Salvador::SSA
Outbound Date: December 20 2009
Inbound Date: January 10 2010

Search Flights

Figure 5: A View generated from a Model in our proposed method.

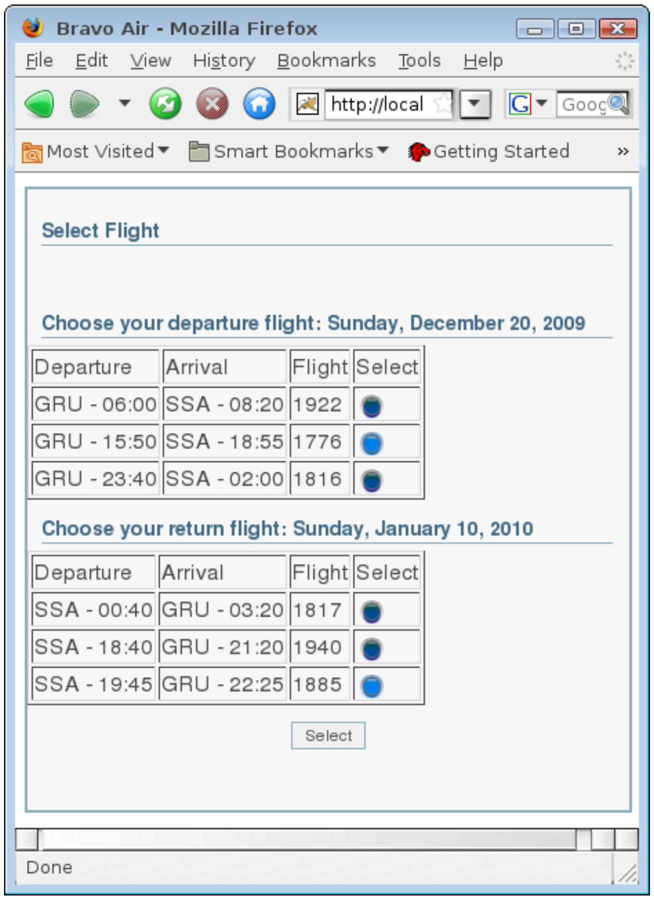
Schemas from XMI descriptions [10], while in Gasevic et al. propose an approach to generate OWL from a UML model [14].

From the perspective of Semantic Web application engineering, efforts involve providing frameworks and infrastructures to support building applications starting from formally defined ontologies [9, 18, 25].

Several researchers investigate approaches to model Semantic Web Services by using UML diagrams. Ha and Lee [17] use UML to model Semantic Web Service applications and to map from OWL-S classes to WSDL abstract types in XML Schemas. Acuna and Marcos [1] use a case study to present an approach based on Model-Driven Architecture (MDA)[h] for developing Semantic Web Services. Barret et al. [5] combine MDA in UML 2.0 for modeling and generating Web Services compositions. Bauer and Muller [6] also use MDA as the starting point for develop Semantic Web Services. They automatically transform UML 2.0 sequence diagrams in a Web Service composition language representation. Kapitsaki et al. [19] use MDA to automatically generate compositions of user-centric web applications in the MVC architecture.

Other authors [20, 24, 32, 36], like those whose work is detailed in Section 3, investigate the use of UML diagrams for the automatic generation of Semantic Web Services descriptions, as well as support to allow transformations both ways between UML and OWL-S [15]. In this paper we may assume the use of one of these approaches to obtain OWL-S descriptions as a

[h]http://www.omg.com/mda


Bravo Air - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://local
Most Visited Smart Bookmarks Getting Started
Done

Select Flight

Choose your departure flight: Sunday, December 20, 2009

Departure	Arrival	Flight	Select
GRU - 06:00	SSA - 08:20	1922	●
GRU - 15:50	SSA - 18:55	1776	●
GRU - 23:40	SSA - 02:00	1816	●

Choose your return flight: Sunday, January 10, 2010

Departure	Arrival	Flight	Select
SSA - 00:40	GRU - 03:20	1817	●
SSA - 18:40	GRU - 21:20	1940	●
SSA - 19:45	GRU - 22:25	1885	●

Select

Figure 6: A View generated from a Model in our proposed method.

starting point to our OWLS2MVC processing.

Several works relate MVC and Web applications [7, 8, 12, 13, 16, 21, 33], and propose different approaches to integrate several technologies (e.g.Java, .NET, XSLT and XForms) to develop Web applications in MVC architecture. In this work, we also integrate some of these technologies to provide a MVC architecture and we propose that the artifacts be automatically generated from OWL-S descriptions.

Our work is also related to others generating Web applications via XSLT transformations. Yan et al. [35] propose style sheets transformations (using XSLT) to design the Web GUI for instruments in remote labs. Andrade et al. [2, 3] propose a document-based approach to generate Web applications based on the application of successive models derived from separate design concerns: conceptual model (using XMI), navigational model and presentation model. Macedo et al. report the use of XSLT transformations [26] in the automatic generation of multimedia documents as Web applications. Similarly, we use XSLT transformations to obtain the Views of our MVC architecture.

7 Final Remarks

Built upon the OWL, the Web Ontology Language, the Ontology Web Language for Services OWL-S has been designed to support the discovery, the selection, the invocation and the composition of Web Services.

Considering the substantial learning curves for OWL and for OWL-S in particular, the literature reports several efforts to support building such specifications higher level models such as UML diagrams. However, an even higher level of automation can be achieved if complete Web applications could be generated from OWL-S specifications. Towards this end, we have presented an approach for processing OWL-S descriptions through a series of transformations that produce MVC-based skeletons for Web applications. Using a starting point a Semantic Web Service modeled with UML diagrams, the overall processing we propose is general, and in the presentation in this paper we extend our original presentation [31] by providing further details of the generation of the Model-View-Controller structure into JavaBeans/JSP/Servlets code.

There are several paths to be taken to continue the work. We have discussed elsewhere parallel efforts toward designing WebLabs as Semantic Web Services [28], as well toward investigating problems associated with the construction of applications presenting temporal restrictions [30], as it is the case in most WebLabs. Our current efforts investigate using the OWLS2MVC approach to built WebLabs applications from their UML and OWL-S models.

Acknowledgments

We would like to thank FAPESP for supporting the authors in the project in which context this work has been developed. We also thank the following organizations: CAPES, for financial support to Cássio Prazeres; FINEP, CNPq, CAPES, RNP and the MCT for the financial support to Maria da Graça Pimentel.

References

1. C. J. Acuna and E. Marcos. Modeling semantic web services: a case study. In *ICWE '06: Proc. of the International Conference on Web Engineering*, pages 32–39. ACM, 2006.
2. A. R. Andrade, E. V. Munson, and M. G. Pimentel. A document-based approach to the generation of web applications. In *DocEng '04: Proceedings of the 2004 ACM symposium on Document engineering*, pages 45–47. ACM, 2004.
3. A. R. Andrade, E. V. Munson, and M. G. Pimentel. Engineering web applications with XML and XSLT. In *LA-WEBMEDIA '04: Proceedings of the WebMedia & LA-Web 2004 Joint Conference 10th Brazilian Symposium on Multimedia and the Web 2nd Latin American Web Congress*, pages 86–93. IEEE Computer Society, 2004.
4. L. Baresi, P. Fraternali, and G.-J. Houben, editors. *Web Engineering, 7th International Conference, ICWE 2007, Como, Italy, July 16-20, 2007, Proceedings*, volume 4607 of *Lecture Notes in Computer Science*. Springer, 2007.
5. R. Barrett, L. M. Patcas, C. Pahl, and J. Murphy. Model driven distribution pattern design for dynamic web service compositions. In *ICWE '06: Proceedings of the 6th international conference on Web engineering*, pages 129–136. ACM, 2006.
6. B. Bauer and J. P. Muller. Mda applied: From sequence diagrams to web service choreography. In *ICWE '04: Proceedings of the 4th international conference on Web engineering*, volume 3140 of *LNCS*, pages 132–136. Springer-Verlag, July 2004.
7. M. Brambilla and A. Origgi. MVC-Webflow: An AJAX Tool for Online Modeling of MVC-2 Web Applications. In *ICWE'08: Eighth International Conference on Web Engineering*, pages 344–349,

July 2008.

8. R. Cardone, D. Soroker, and A. Tiwari. Using XForms to simplify Web programming. In *WWW '05: Proceedings of the 14th international conference on World Wide Web*, pages 215–224. ACM, 2005.
9. O. Corcho, A. López-Cima, and A. Gómez-Pérez. A platform for the development of semantic web portals. In *ICWE '06: Proceedings of the 6th international conference on Web engineering*, pages 145–152. ACM, 2006.
10. S. Cranefield. Networked Knowledge Representation and Exchange using UML and RDF. *Journal of Digital Information*, 1(8):http://journals.tdl.org/jodi/article/view/30/31, 2001.
11. M. Dean, G. Schreiber, S. Bechhofer, F. van Harmelen, J. Hendler, I. Horrocks, D. L. McGuinness, P. F. Patel-Schneider, and L. A. Stein. OWL Web Ontology Language Reference – W3C Recommendation 10 February 2004, February 2004.
12. D. Distante, P. Pedone, G. Rossi, and G. Canfora. Model-Driven Development of Web Applications with UWA, MVC and JavaServer Faces. In Baresi et al. [4], pages 457–472.
13. F. J. García, R. I. Castanedo, and A. A. J. Fuente. A Double-Model Approach to Achieve Effective Model-View Separation in Template Based Web Applications. In Baresi et al. [4], pages 442–456.
14. D. Gasevic, D. Djuric, and V. D. Vladan. Mda-based automatic owl ontology development. *Int. J. Softw. Tools Technol. Transf.*, 9(2):103–117, 2007.
15. R. Grønmo, M. C. Jaeger, and H. Hoff. Transformations between uml and owl-s. In *Proceedings of the First European Conference on Model Driven Architecture - Foundations and Applications, (ECMDA-FA 2005)*, pages 269–283, 2005.
16. L. GuangChun, W. Lu, and X. Hanhong. A novel web application frame developed by MVC. *SIGSOFT Softw. Eng. Notes*, 28(2):7, 2003.
17. Y. Ha and R. Lee. Semantic Web Service Modeling using UML for e-business environment. In *SNPD-SAWN '06: Proceedings of the Seventh ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing*, pages 368–374, Washington, DC, USA, 2006. IEEE Computer Society.
18. Y. Jin, S. Decker, and G. Wiederhold. Ontowebber: Model-driven ontology-based web site management. In *Proceedings of SWWS'01, The First Semantic Web Working Symposium*, pages 529–547 http://infolab.stanford.edu/pub/gio/2001/Ontowebber01.pdf, 2001.
19. G. M. Kapitsaki, D. A. Kateros, C. A. Pappas, N. D. Tselikas, and I. S. Venieris. Model-driven development of composite web applications. In *iiWAS '08: Proceedings of the 10th International Conference on Information Integration and Web-based Applications & Services*, pages 399–402. ACM, 2008.
20. I.-W. Kim and K.-H. Lee. Describing semantic web services: From UML to OWL-S. In *ICWS '07: Proceedings of the IEEE International Conference on Web Services*, pages 529–536, Korea, July 2007. IEEE CS Press.
21. S. Kojarski and D. H. Lorenz. Domain driven web development with webjinn. In *OOPSLA '03: Companion of the 18th annual ACM SIGPLAN Conference on Object-oriented Programming, Systems, Languages, and Applications*, pages 53–65. ACM, 2003.
22. M. Konyk, A. D. L. Battista, and M. Dumontier. Chemical Knowledge for the Semantic Web. In *DILS*, pages 169–176, 2008.
23. G. E. Krasner and S. T. Pope. A cookbook for using the model-view controller user interface paradigm in Smalltalk-80. *J. Object Oriented Program.*, 1(3):26–49, 1988.
24. C. Lee, J. Kim, J. Lee, and B. Lee. Evolving web service applications using UML and OWL-S. In *ICCIT '07: Proceedings of the 2007 International Conference on Convergence Information Technology*, pages 1247–1252, Washington, DC, USA, 2007. IEEE Computer Society.
25. A. López-Cima, Ó. Corcho, and A. Gómez-Pérez. Rapid Ontology-based Web Application Development with JSTL. In *Proceedings of the (SFSW 2007) Workshop on Scripting for the Semantic Web (held with ESWC'07)*, 2007.
26. A. A. Macedo, L. Baldochi, J. A. C. Guerrero, R. G. Cattelan, and M. da Graça Campos Pimentel. Automatically linking live experiences captured with a ubiquitous infrastructure. *Multimedia Tools*

Appl., 37(2):93–115, 2008.

27. D. Martin, M. Burstein, J. Hobbs, O. Lassila, D. McDermott, S. McIlraith, S. Narayanan, M. Paolucci, B. Parsia, T. Payne, E. Sirin, N. Srinivasan, and K. Sycara. W3C member submission 22 november 2004 – OWL-S: Semantic markup for web services, November 2004.
28. C. V. S. Prazeres, M. G. C. Pimentel, and C. A. C. Teixeira. Remote experiments as semantic web services. In H. Yu, M. Naphade, and H. Koiti, editors, *ICSC '07: Proceedings of the 1st IEEE International Conference on Semantic Computing*, pages 791–798. IEEE CS Press, Set 2007.
29. C. V. S. Prazeres and C. A. C. Teixeira. A structured document-based approach for WebLab configuration. In *WebMedia '06: Proceedings of the 12th Brazilian symposium on Multimedia and the web*, pages 1–10, Natal, Rio Grande do Norte, Brazil, 2006. ACM Press.
30. C. V. S. Prazeres, C. A. C. Teixeira, and M. da Graça Campos Pimentel. Semantic Web services discovery by matching temporal restrictions. In *SAINT '08: Proceedings of the 8th IEEE/IPSJ International Symposium on Applications and the Internet*, pages 26–32. IEEE Computer Society, 2008.
31. C. V. S. Prazeres, C. A. C. Teixeira, E. V. Munson, and M. da Graça C. Pimentel. Semantic web services: from OWL-S via UML to MVC applications. In *SAC '09: Proceedings of the 2009 ACM symposium on Applied Computing*, pages 675–680. ACM, 2009.
32. J. T. E. Timm and G. C. Gannod. A model-driven approach for specifying semantic web services. In *ICWS '05: Proceedings of the IEEE International Conference on Web Services*, pages 313–320, Washington, DC, USA, 2005. IEEE Computer Society.
33. K. Watanabe, M. Imamura, K. Asami, and T. Amanuma. *A Web Application Development Framework Using Code Generation from MVC-Based UI Model*, volume 5518/2009 of *Distributed Computing, Artificial Intelligence, Bioinformatics, Soft Computing, and Ambient Assisted Living*, chapter 11, pages 404–411. Springer Berlin - Heidelberg, June 2009.
34. A. Z. Wyner. An ontology in OWL for legal case-based reasoning. *Artif. Intell. Law*, 16(4):361–387, 2008.
35. Y. Yan, Y. Liang, and X. Du. Controlling remote instruments using web services for online experiment systems. In *ICWS '05: Proceedings of the IEEE International Conference on Web Services (ICWS'05)*, pages 725–732, Washington, DC, USA, 2005. IEEE Computer Society.
36. J. H. Yang and I. J. Chung. Automatic generation of service ontology from uml diagrams for semantic web services. In *ASWC '06: Proceedings of the First Asian Semantic Web Conference*, volume 4185 of *LNCS*, pages 523–529. Springer, 2006.