Semantic Web Services: from OWL-S via UML to MVC Applications

Cássio V. S. Prazeres
Universidade de São Paulo
prazeres@icmc.usp.br

Cesar A. C. Teixeira
Universidade Federal de São Carlos
cesar@dc.ufscar.br

Ethan V. Munson
University of Wisconsin-Milwaukee
munson@uwm.edu

Maria da Graça C. Pimentel
Universidade de São Paulo
mgp@icmc.usp.br

ABSTRACT

OWL-S is used to describe the semantics of Web Services so that the discovery, selection, invocation and composition of these services can be automated. Prior research has shown that UML diagrams can be used to automatically generate Semantic Web Service descriptions in OWL-S. If complete Web applications could be generated from OWL-S descriptions, then a higher level of automation would be achieved. In this paper, we propose an approach for processing OWL-S descriptions in order to produce MVC-based skeletons for Web applications. The OWL-S ontology goes through a series of transformations in order to generate an application whose Model-View-Controller structure is implemented by a combination of JavaBeans/JSP/Servlets code, respectively.

Category: D.2.2 Design Tools and Techniques - Object-oriented design methods.

General Terms: Design, Documentation, Languages.

Keywords: Design, Implementation, MVC.

1. INTRODUCTION

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” [11]. However, the deployment of semantic-rich description languages such as OWL-S is not trivial [15], 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” [11].

Given the substantial learning curve for OWL-S, some researchers have proposed that Semantic Web Services descriptions be generated automatically from UML diagrams (e.g. [7]). 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.

The Model-View-Controller (MVC)\(^1\) architectural pattern, originally deployed\(^2\) in Smalltalk [9], is widely used in the construction of Web applications (e.g. [4, 6, 8]). We propose an approach for producing MVC-based skeletons for Web applications from OWL-S descriptions. 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)\(^3\), 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 detailing the generation of the JavaBeans, Servlets and JSP code for a fictitious airline site that is a classical example in the literature of OWL-S.

In this paper, Section 2 describes approaches for generating Semantic Web Services Descriptions from UML diagrams; Section 3 reviews the core concepts of OWL-S; Section 4 revisits our work exploiting XSL Transformations from XML Schemas to Web-based user interfaces; Section 5 details the generation of MVC Web applications from OWL-S descriptions; Section 6 reviews related work and Section 7 concludes the paper.

2. FROM UML TO OWL-S

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

\(^1\)http://heim.ifi.uio.no/~trygver/themes/mvc/mvc-index.html

\(^2\)http://st-www.cs.uiuc.edu/users/smarch/st-docs/mvc.html

\(^3\)http://www.omg.org/technology/documents/formal/xmi.htm
applications [1, 2, 3], and (b) research that uses UML diagrams to automatically generate Semantic Web Services descriptions [7, 10, 15, 17]. It is the second type that is particularly relevant to the research presented in this paper.

Kim and Lee [7] 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 [17] 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 and that information about composition of services could be extracted from UML statechart diagrams. Lee et al. [10] 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 [15] 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 [7]. These OWL-S specifications are then used to generate MVC-structured Web applications.

3. SEMANTIC WEB SERVICES

The OWL-S upper ontology for services aims to provide three essential types of knowledge about a service: the ServiceProfile, the ServiceModel and the ServiceGrounding [11]. In OWL-S, the knowledge about a service must be formally described by the developer as follows: (a) the class ServiceProfile must describe “what the service does” by formalizing the inputs, outputs, preconditions and effects of the service; (b) the class ServiceModel describes “how the service works” by formalizing the composition of the service based on processes and control constructs such as sequences; (c) the class ServiceGrounding describes “how to access the service” by detailing information such as communication protocols (e.g. SOAP) and message formats (e.g. WSDL) [11].

The ServiceProfile provides a superclass of every type of high-level description of the service and is used for publishing (providers) and discovering (requesters) 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, which forms 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.

The Process class includes three types of processes: Atomic, Composite and Simple. Atomic processes are directly invo- 

able (by passing them the appropriate messages), have no subprocesses, and from the perspective of the service requester, execute in a single step. 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. 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 [11].

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

4. FROM XML SCHEMAS TO WEB-BASED USER INTERFACES VIA XSLT

In order to generate the MVC-Views (see Section 5.4) for Web applications in the MVC architecture, the approach proposed in this paper builds upon our previous work [14], in which we propose the use of XSLT style sheets transformations to the design of Web-based interfaces to WebLabs. WebLabs are laboratories that allow resources to be remotely accessed by means of experiments controlled via a computer network. Considering the variety of WebLabs and experiments that can be defined, our approach has been designed to facilitate the extension and customization of Web-based applications according to WebLabs requirements. This has been achieved by intensive use XML Schemas for structuring the information defined not only a priori, considering the WebLabs application domain, but also on-the-fly, to accommodate the unique characteristics of each lab.

The support to 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 detailed in Fig. 1.

Figure 1: Document-based flow for WebLab registration

In Fig. 1, 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 filled by the WebLab administrator with the corresponding information from his lab. Once the form is filled in, the corresponding lab specification is validated (to guarantee conformance with datatypes and structure). More specifically, in Fig. 1:

- the XML Schema is indicated by WLC: Web Lab Concept;

OWL-S classes for Web Services inputs, outputs, preconditions and effects.

676
• 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 filling of the form by the actor 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 actor 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 in Document 1 illustrates, as an example, the type of information demanded to be registered for each lab (WLC: Web Lab Concept in Fig. 1). The processing of this document with an appropriate transformation (WLC2FORM in Fig. 1) generates the form presented in Fig. 2 (WLDF in Fig. 1).

The processing detailed in this section illustrates the use of XSLT style sheets transformations to the design of Web-based interfaces to WebLabs. We built upon this transformation style to process OWL-S specifications to generate the MVC-View component as the JSP code of the target Web application.

5. OWLS2MVC: FROM OWL-S TO AN MVC-BASED WEB APPLICATION

The core business model functionality is separated from the presentation and control logic in a MVC architecture. Such architecture allows the same enterprise data model having multiple views\(^5\). The goal is to decouple the graphical interface of navigation and behavior of the application. This decoupling promotes easier maintenance and greater reuse.

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

5.1 The OWLS2MVC processing

Fig. 3 presents the workflow generating MVC-based Web applications from OWL-S descriptions.

---

\(^5\)http://java.sun.com/blueprints/patterns/MVC-detailed.html
In the second step, the data from OWL-S descriptions is extracted using the OWL-S API\(^6\), which generates two documents: the Servlets that implements the MVC Controller (Fig. 3(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 (Fig. 3(b)).

In the fourth step, the generation of JSP code (which results in the rendering of HTML documents) maps to the View components of the MVC architecture (Fig. 3(c)).

Sections 5.2, 5.3 and 5.4 describe how each file of the MVC architecture is generated using the classical example of OWL-S: the Bravo Air\(^1\) – a fictitious airline site.

5.2 Model

In OWL2S2MVC, a MVC Model is obtained from the inputs and outputs of a service in OWL-S. 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, an XML Schema is generated and the JavaBeans for the Model are created (see Fig. 3).

Document 2 shows examples of inputs for an OWL-S process. For example, the input DepartureAirport\(_{\text{in}}\) at lines 3 to 5 in Document 2 is the departure airport for the Bravo Air service.

```xml
1 <!-- Inputs for a process in OWL-S -->
2 <process:Input rdf:ID="DepartureAirport\_in"/>
3 <process:Input rdf:ID="ArrivalAirport\_in"/>
4 <process:Input rdf:ID="FlightDate"/>
5 </process:Input>
```

These inputs are defined by using XML Schemas as in Document 3. The Airport\(_{\text{class}}\) (lines 3 to 5 in Document 3) defines the type of the input DepartureAirport\(_{\text{in}}\) (lines 3 to 5 in Document 2), and it has the datatype string defined in XML Schema datatypes\(^8\).

```xml
1 <!-- Types definition for the Inputs -->
2 <owl:Class rdf:ID="Airport">
3 <rdfs:subClassOf rdf:resource="&xsd;#string"/>
4 </owl:Class>
```

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.

```java
1 public class DesiredFlightBean {
2 private boolean roundTrip_In;
3 private Date inboundDate_In;
4 private Date outboundDate_In;
5 private String departureAirport_In;
6 private String arrivalAirport_In;
7 private Date outboundDate_In;
8 private Date inboundDate_In;
9 private boolean roundTrip_In;
10 public DesiredFlightBean(String departureAirport_In) {
11 this.departureAirport_In = departureAirport_In;
12 return this;
13 }
14 public void setDepartureAirport_In(String departureAirport_In) {
15 this.departureAirport_In = departureAirport_In;
16 }
17 public String getDepartureAirport_In() {
18 return departureAirport_In;
19 }
20 public DesiredFlightBean(Date outboundDate_In) {
21 this.outboundDate_In = outboundDate_In;
22 return this;
23 }
24 public void setOutboundDate_In(Date outboundDate_In) {
25 this.outboundDate_In = outboundDate_In;
26 }
27 public Date getOutboundDate_In() {
28 return outboundDate_In;
29 }
30 public DesiredFlightBean(Date inboundDate_In) {
31 this.inboundDate_In = inboundDate_In;
32 return this;
33 }
34 public void setInboundDate_In(Date inboundDate_In) {
35 this.inboundDate_In = inboundDate_In;
36 }
37 public boolean isRoundTrip_In() {
38 return roundTrip_In;
39 }
40 public void setRoundTrip_In(boolean roundTrip_In) {
41 this.roundTrip_In = roundTrip_In;
```

5.3 Controller

The fourth step of our OWL2S2MVC flow generates Java files that are Servlet classes. These Servlets are the Controller components in our MVC architecture. The Controller is generated by extracting information about the composite processes from the OWL-S specification.

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: the Servlet Controller. Atomic processes that compose the composite process are mapped to methods inside the Servlet Controller. The workflow of each Servlet is generated by the control constructs (Such as If-Then-Else and Repeat-While detailed in...
Table 1) of OWL-S. The Models, generated as explained in Section 5.2, are arguments of the Java method in the Servlet Controller.

Table 1: OWLS to Java Mapping

<table>
<thead>
<tr>
<th>Type</th>
<th>OWL-S</th>
<th>Java Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Process</td>
<td>composite</td>
<td>Servlet Class</td>
</tr>
<tr>
<td>atomic</td>
<td>Method</td>
<td>Method</td>
</tr>
<tr>
<td>Control Construct</td>
<td>Sequence</td>
<td>sequence of code</td>
</tr>
<tr>
<td>If-Then-Else</td>
<td>if-then-else</td>
<td></td>
</tr>
<tr>
<td>Choice</td>
<td>switch</td>
<td>Method</td>
</tr>
<tr>
<td>Repeat-While</td>
<td>while</td>
<td>Method</td>
</tr>
<tr>
<td>Repeat-Until</td>
<td>do-while</td>
<td></td>
</tr>
<tr>
<td>Parameter</td>
<td>Input</td>
<td>arguments of Method</td>
</tr>
<tr>
<td>parameterType</td>
<td>type of attributes</td>
<td></td>
</tr>
<tr>
<td>Output</td>
<td>return of Method</td>
<td></td>
</tr>
</tbody>
</table>

Document 5 presents a composite process in OWL-S. This process is composed of a sequence of two atomic process (lines 8 and 9 in Document 5) and a composite process (line 10 in Document 5). This composite process is mapped into a Servlet Controller as shown in Document 7.

Document 6 presents the definition of the atomic process GetDesiredFlightDetails. This process is one of the three processes that form the composite process shown in Document 5. Lines 4 to 8 of Document 6 describe the inputs for the atomic process. Each atomic process will be mapped as a method in the Servlet Controller as shown in Document 7 (lines 23 to 27).

Document 6 describes a composite process as a sequence of three process (two atomics and one composite). This sequence is mapped for 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 7). Lines 23 to 27 in Document 7 correspond to the atomic process in the Servlet Controller: getDesiredFlightDetails and selectAvailableFlight.

As illustrated in Document 7 (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 5 (line 10). The BookFlight Servlet, which is not detailed in this paper, should be mapped to another Servlet Controller using the same process described in this section.

5.4 View

Fig. 4 shows an example of one of many possibilities of automatically generated Views. The Web developer can modify this View or create other Views from scratch. The Views are generated in the fourth step of our proposed flow. 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. The form of Fig. 4 corresponds to the Model presented in Document 4.

All Views can be generated by using the approach presented in Section 4. In order to generate the View of Fig. 4, the XML Schema datatypes of Document 3 pass by a trans-

Other authors, as those whose work is detailed in Section 2 [7, 17, 10, 15], investigate the use of UML diagrams to the automatic generation of Semantic Web Services descriptions. In this paper we assume the use one of these approaches to obtain OWL-S descriptions as a starting point to our OWLS2MVC processing.

Several works relate MVC and Web applications [6, 8, 4], 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. [16] propose style sheets transformations (using XSLT) to design the Web GUI for instruments in remote labs. Andrade et al. [5] 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. Similarly, we use XSLT transformations to obtain the Views of our MVC architecture.

6. RELATED WORKS


Other authors, as those whose work is detailed in Section 2 [7, 17, 10, 15], investigate the use of UML diagrams to the automatic generation of Semantic Web Services descriptions. In this paper we assume the use one of these approaches to obtain OWL-S descriptions as a starting point to our OWLS2MVC processing.

Several works relate MVC and Web applications [6, 8, 4], 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. [16] propose style sheets transformations (using XSLT) to design the Web GUI for instruments in remote labs. Andrade et al. [5] 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. Similarly, we use XSLT transformations to obtain the Views of our MVC architecture.

7. FINAL REMARKS

In this paper, we have proposed OWLS2MVC as an approach to produce MVC-based skeletons for Web applications from OWL-S descriptions.

We have discussed elsewhere parallel efforts toward making WebLabs as Semantic Web Services [12], as well toward investigating problems associated with the construction of applications presenting temporal restrictions [13], as it is the case in most WebLabs. Among our targets for future work, we plan to deploy the OWLS2MVC approach to built WebLabs applications from their UML and OWL-S models, which are to be integrated via yet other semantic services.

Acknowledgments We thank the support from FAPESP, CAPES, CNPq and FINEP.

8. REFERENCES