Interface Development for Hypermedia Applications in the Semantic Web

Sabrina Silva de Moura, Daniel Schwabe
Departamento de Informática, PUC-Rio
{sabrina, dschwabe}@inf.puc-rio.br

Abstract

The Semantic Web has been gaining increasing attention, spurring a large number of initiatives to design and implement applications in this environment. This paper proposes an approach to specifying the user interface to such applications, as part of the Semantic Hypermedia Design Method. It proposes the use of an Abstract Interface Ontology, which is mapped onto application elements on one side and onto concrete interface elements specified as instances of a Concrete Interface Ontology. An implementation architecture, based on JSP and TagLibs is also proposed.

1. Introduction

There are several proposals for designing applications (at least, websites) in the so-called Semantic Web, mostly RDF-based ([10][11][12]). More recently, Web Engineering approaches have been proposed for designing and implementing such applications, such as the Semantic Hypermedia Design Method (SHDM) [14][16]. This methodology applies the experience gathered with OOHDM [18] in the context of the Semantic Web, where it tries to leverage the formalisms proposed for its foundation, such as RDF [13] and OWL [19][20].

One aspect that has received little attention is the leveraging of these formalisms to the design and specification of application interfaces. By design here, we mean logical design, as opposed to layout and appearance (graphical) design.

In this paper, we present an approach for designing and specifying application interfaces for the Semantic Web, as part of SHDM. This approach can be called semantic for two reasons. The first is because the interface is described at the level of abstraction of information exchanges between the user and the application, therefore closer to the task being performed. A successive step defines the layout and appearance.

The second, perhaps more arguable, is that we use the formalisms for the Semantic Web, which allow direct manipulation to generate the final application. We outline an implementation architecture that allows this direct interpretation of the semantic specifications.

The organization of the remainder of the paper is as follows. Section 2 presents a summary of SHDM, section 3 presents our proposal for the Interface model; section 4 discusses an implementation architecture, section 5 discusses related work and section 6 draws some conclusions and points to future work.

2. SHDM Summary

SHDM is a model-driven approach to design web applications using five different steps: Requirements Gathering, Conceptual Design, Navigational Design, Abstract Interface Design, and Implementation.

Very briefly, an SHDM design sees an application in the Semantic Web as a view (mapping) over some conceptual ontology describing a given problem domain. This view is oriented towards supporting a specific set of tasks performed by a set of users under a certain set of roles. It as starts with a conceptual model of the application domain, described as some ontology using either OWL or RDFS.

Next, this conceptual ontology is mapped onto a navigational ontology, describing the artifacts that will be navigated and manipulated by the users. These operations, as well as the application specific operations (business logic) are actually accessed by the user through an interface that isolates presentation aspects from the application logic (both business logic and navigation).

In order to achieve separation of concerns at the interface level, it is actually split into two parts – the abstract interface, focusing on the information exchange needs between the application and its users, and the concrete interface, focusing on the look and feel, and on the actual runtime environment (both software and hardware aspects). Table 1 lists the artifacts produced by each phase.

Each step focuses on a particular aspect and produces models, describing details about an application to be run on the web.

The separation between conceptual and navigational design is an important cornerstone of OOHDM that was kept in SHDM. By explicitly separating conceptual from navigation design, we address different concerns in web applications. Whereas conceptual modeling and design must reflect objects and behaviors in the application

domain, navigation design aims at organizing the hyperspace, taking into account users' profiles and tasks.

Table 1. SHDM artifacts

	Artifact	Definition Language	Description
1	Conceptual Ontology	OWL-DL with annotations and addtional SHDM rules	Conceptual class definitons
2	Conceptual instances	Conceptual Ontology	Application data defined according to the Conceptual Ontology.
3	Navigational mapping	Navigational mapping definition vocabulary	Rules mapping conceptual classes into navigational classes.
4	Navigation space definition	Navigation space definition vocabulary	Definition of the navigational elements – contexts and access structures (indexes).
5	Navigational Ontology	OWL-DL	Navigational class (node) definitions.
6	Navigational instances	Navigational Ontology	Application data defined according to the Navigational Ontology.
7	Abstract Interface	Abstract Interface definition vocabulary	Abstract interface definition, including abstract interface elements and their mapping to the navigation model and to concrete interface widgets.
8	Concrete interface widget ontology	Definition vocabulary for concrete interface widgets	Definition of possible concrete interface widgets to be used in the implementation

Navigational design is a key activity in the implementation of web applications, and we advocate that it must be explicitly separated from conceptual modeling. In SHDM, the navigational design step produces expressive models capable of representing web applications, and even families of web applications.

The examples in the following sub-sections will help clarify these concepts (we don't include Requirements Gathering in this paper); additional details can be found in [14].

The information items described in the Conceptual Model and in the Navigation Class Schema are resources specified in RDF. The characterization of resources in SHDM is done using OWL, expressing constraints (restrictions), enumeration and XML Schema data types.

The typical workflow in producing these artifacts is (the numbers in brackets refer to the first column in Table 1):

1. Conceptual Ontology design {1}.
2. Once the Conceptual Ontology has been defined, instances {2} can be created at anytime.
3. Navigational mapping definition {3}.
4. Navigational space specification {4}.
5. Once the navigational space has been defined, the Navigational Ontology {5} and the corresponding navigational instances {6} can be automatically generated based on artifacts {1, 2, 3, 4}. It should be noted that artifacts 5 and 6 need only be actually materialized, instead of dynamically computed, for optimization purposes, similarly as in the case of materialized views for databases.
6. Abstract Interface definition {7}.

Notice that artifact 8 is typically pre-defined, culled from existing interface definition languages, and needs updating only when new interface technologies are introduced.

3. Interface Specification in SHDM

As previously outlined, Conceptual design focuses on characterizing the information elements of the application domain, and Navigation design focuses on supporting users in achieving their intended tasks. The Abstract Interface design focuses on making Navigation objects and application functionality perceptible to the user, which must be done at the application interface.

Even while focusing on the interface, it is possible to factor out various design concerns. At the most abstract level, the interface functionality can be thought as supporting information exchange between the application and the user, including activation of functionalities. In fact, from this standpoint, navigation is just another (albeit distinguished) application functionality.

Since the tasks being supported drive this information exchange, it is reasonable to expect that it will be less sensitive to runtime environment aspects, such as particular standards and devices being used. The design of this aspect of the interface can be carried out by interaction designers or software engineers.

At a more concrete level, it is necessary to define the actual look and feel of the application, including layout, font, color, and graphical appearance. Graphics designers typically carry this out. This part of the design is almost totally dependent on the particular hardware and software runtime environment.

Such separation allows shielding a significant part of the interaction design from inevitable technological platform evolution, as well as from the need to support users in a multitude of hardware and software runtime environments.

3.1 Abstract Widget Ontology

The most abstract level is called the Abstract Interface, focusing on the type of functionality played by interface elements. The Abstract Interface is specified using the Abstract Widget Ontology, which establishes the vocabulary, shown in Figure 1.

An abstract interface widget can be any of the following:

- SimpleActivator, which is capable of reacting to external events, such as mouse clicks;
- ElementExhibitor, which is able to exhibit some type of content, such as text or images;

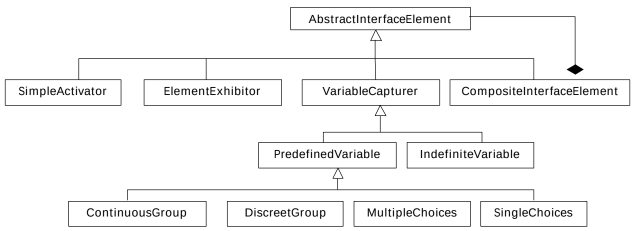

Figure 1. Abstract Widget Ontology

- VariableCapturer, which is able to receive (capture) the value of one or more variables. This includes input text fields, selection widgets such as pull-down menus and checkboxes, etc... It generalizes two distinct (sub) concepts;
- IndefiniteVariable, which allows entering hitherto unknown values, such as a text string typed by the user;
- PredefinedVariable, which stands for widgets that allow the selection of a subset from a set of pre-defined values; oftentimes the selection must be a singleton. Specializations of this concept are ContinousGroup, DiscreetGroup, MultipleChoices, and SingleChoice. The first allows selecting a single value from an infinite range of values; the second is analogous, but for a finite set; the remainder are self-evident.
- CompositeInterfaceElement, which is a composition of any of the above.

It can be seen that this ontology captures the essential roles that interface elements play with respect to the interaction – either they exhibit information, or they react to external events, or they accept information. As customary, composite elements allow building more complex interfaces out of simpler building blocks.

The software designer, who understands the application logic and the kinds of information exchanges that must be supported to carry out the operations, should carry out the abstract interface design. This software designer does not have to worry about usability issues, or look and feel, which will be dealt with during the concrete interface design, normally carried out by a graphics (or "experience") designer.

Once the Abstract Interface has been defined, each element must be mapped onto both a navigation element, which will provide its contents, and a concrete interface widget, which will actually implement it in a given runtime environment. Figure 2 shows an example of an interface for an academic website, and Figure 3 shows an abstract representation of this interface.

Before proceeding to show how this is achieved, we must first define the Concrete Widget ontology, which characterizes the actual widgets available in concrete runtime environments.

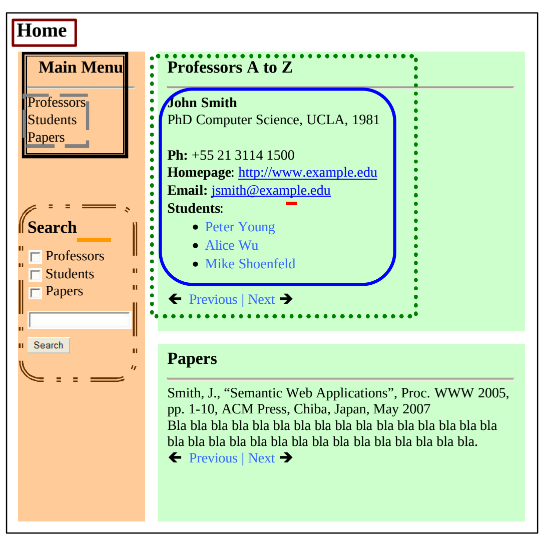

Figure 2. An example concrete interface

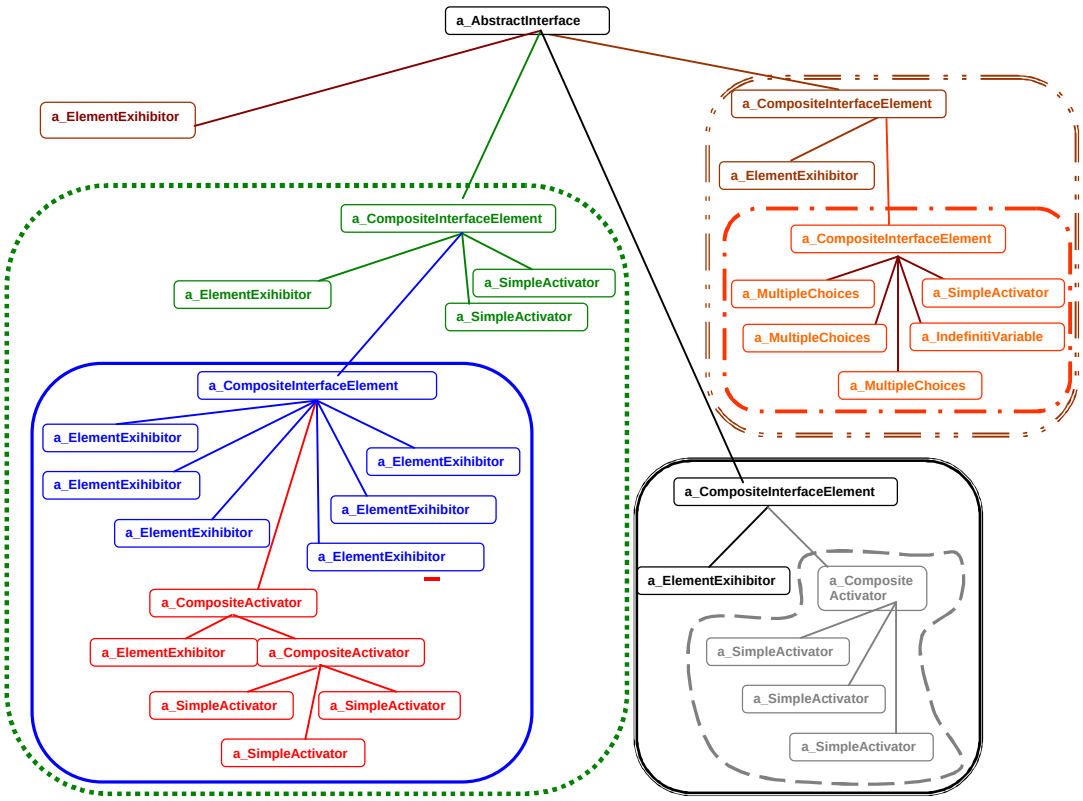

Figure 3. Abstract widget ontology instance for the example in Figure 2.

3.2 Concrete Widget Ontology

The purpose of this ontology is to describe actual widgets commonly available in most graphical interface runtime environments, such as XulPlanet [5], Java Swing [8] , User Interface Markup Language (UIML) [1] and the PIMA project at IBM [3] among others. This ontology, shown in Figure 4, can be extended as new widgets appear in currently available environments.

```
<ConcreteInterfaceElement rdf:ID="Button" />
<ConcreteInterfaceElement rdf:ID="CheckBox" />
<ConcreteInterfaceElement rdf:ID="CheckBoxAction" />
<ConcreteInterfaceElement rdf:ID="ComboBox" />
<ConcreteInterfaceElement rdf:ID="ComboBoxAction" />
<ConcreteInterfaceElement rdf:ID="ComboBoxTarget" />
<ConcreteInterfaceElement rdf:ID="Composition" />
<ConcreteInterfaceElement rdf:ID="Form" />
<ConcreteInterfaceElement rdf:ID="Image" />
<ConcreteInterfaceElement rdf:ID="Label" />
<ConcreteInterfaceElement rdf:ID="Link" />
<ConcreteInterfaceElement rdf:ID="RadioButon" />
<ConcreteInterfaceElement rdf:ID="RadioButonAction" />
<ConcreteInterfaceElement rdf:ID="RadioButonTarget" />
<ConcreteInterfaceElement rdf:ID="TextArea" />
<ConcreteInterfaceElement rdf:ID="TextBox" />
```

Figure 4. Concrete Widget Ontology

In this ontology, the names are self-describing. CheckBoxAction, ComboBoxAction and RadioButtonAction correspond to concrete widgets in which the selection of the element executes the submit action, without need for an additional confirmation step on the user's part.

It should be stressed that this concrete ontology, as it stands, is very superficial. In particular, is not our immediate goal to synthesize concrete interfaces automatically, but rather to allow the designer to make the choices. For this reason, we have not attempted to describe in detail all the constraints that real widgets must satisfy, or the properties that would help to derive a concrete design in a fully automated way.

Therefore, for our purposes, the concrete widget ontology is used simply to record the necessary information to allow the generation of the JSP page and the corresponding TagLibs, as discussed later on.

3.3 Mappings

The Abstract Interface Ontology actually contains, for each abstract interface widget, both the mapping to navigation (i.e., application specific) elements, and to a concrete interface element.

The *mapsTo* property, which is an *ObjectProperty*, represents this, as can be seen in Figure 5.

```
...
<!DOCTYPE rdf:RDF [
        <!ENTITY cwo  "http://www.inf.puc-rio.br/~sabrina/ontology/CW/cwo#" >
        <!ENTITY awo  "http://www.inf.puc-rio.br/~sabrina/ontology/AW/awo#" >
]>

<rdf:RDF xmlns:cwo="&cwo;"xmlns:awo="&awo;"
                        ...

<owl:ObjectProperty rdf:ID="mapsTo">
        <rdfs:range rdf:resource="&cwo;ConcreteInterfaceElement" />
        <rdfs:domain rdf:resource="&awo;AbstractInterfaceElement" />
 </owl:ObjectProperty>
```

Figure 5. Definition of the *mapsTo* property.

There is additional information in the ontology restricting each abstract interface widget to compatible concrete interface widgets, as illustrated in Figure 6. This snippet states that the "SimpleActivator" abstract interface widget can only be mapped into the "Link" or "Button" concrete interface widgets.

```
- <owl:Class rdf:ID="CompositeInterfaceElement">
  - <rdfs:subClassOf>
    - <owl:Restriction>
      - <owl:allValuesFrom>
        - <owl:Class>
          - <owl:oneOf rdf:parseType="Collection">
              <cwo:ConcreteInterfaceElement rdf:about="http://www.inf.puc-
                rio.br/~sabrina/ontology/CWO/cwo#Form" />
              <cwo:ConcreteInterfaceElement rdf:about="http://www.inf.puc-
                rio.br/~sabrina/ontology/CWO/cwo#Composition" />
            </owl:oneOf>
          </owl:Class>
        </owl:allValuesFrom>
      - <owl:onProperty>
          <owl:ObjectProperty rdf:about="#mapsTo" />
        </owl:onProperty>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf rdf:resource="#AbstractInterfaceElement" />
  </owl:Class>
```

Figure 6. Mapping restrictions for abstract and concrete interface widgets.

Actual abstract interface widget instances are mapped onto specific navigation elements (in the navigation ontology) and onto concrete interface widgets (in the Concrete Interface Widget Ontology). Figure 7 shows the specification of the "Previous Professor" (of class "SimpleActivator") abstract interface widget shown in Figure 2, which is mapped onto a "Link" concrete interface element.

```
<awo:SimpleActivator rdf:ID="ProfessorAlphaPrevious">
  <awo:fromElement>ctxProfessorAlpha</awo:fromElement>
  <awo:fromAttribute>_PREV</awo:fromAttribute>
  <awo:abstractInterface>ProfessorAlpha</awo:abstractInterface>
  <awo:mapsTo rdf:resource="www.tecweb.inf.puc-
    rio.br:8080/ontology/CWO/cwo#Link" />
</awo:SimpleActivator>
```

Figure 7. Mapping between abstract interface widget and navigation element.

Figure 8 shows an example illustrating how application functionality is integrated, giving the OWL specification of the "Search" abstract interface element. It is composed of two abstract widgets, "ElementExhibitor" (lines 9-12), and "CompositeInterfaceElement" (lines 14-46). The first shows the "Search" string, using a "Label" concrete widget. The second aggregates the four elements used to specify the field in which the search may be performed, namely, three "MultipleChoices" – SearchProfessors (lines 25-29), SearchStudents (31-35) e SearchPapers (37-41) - and one "IndefiniteVariable" – "SearchField" (lines 43-46).

```
                                        ...
1   <awo:CompositeInterfaceElement rdf:ID="Search">
2       <awo:fromIndex>idxSearch</awo:fromIndex>
3       <awo:mapsTo rdf:resource="&cwo;Composition"/>
4       <awo:isRepeated>false</awo:isRepeated>
5       <awo:hasInterfaceElement rdf:resource="#TitleSearch"/>
6       <awo:hasInterfaceElement rdf:resource="#SearchElements"/>
7   </awo:CompositeInterfaceElement>
8
9   <awo:ElementExihibitor rdf:ID="TitleSearch">
10      <awo:visualizationText>Search</awo:visualizationText>
11      <awo:mapsTo rdf:resource="&cwo;Label"/>
12  </awo:ElementExihibitor>
13
14  <awo:CompositeInterfaceElement rdf:ID="SearchElements">
15      <awo:fromIndex>idxSearch</awo:fromIndex>
16      <awo:abstractInterface>SearchResult</awo:abstractInterface>
17      <awo:mapsTo rdf:resource="&cwo;Form"/>
18      <awo:isRepeated>false</awo:isRepeated>
19      <awo:hasInterfaceElement rdf:resource="#SearchProfessors"/>
20      <awo:hasInterfaceElement rdf:resource="#SearchStudents"/>
21      <awo:hasInterfaceElement rdf:resource="#SearchPapers"/>
22      <awo:hasInterfaceElement rdf:resource="#SearchField"/>
23  </awo:CompositeInterfaceElement>
24
25  <awo:MultipleChoices rdf:ID="SearchProfessors">
26      <awo:fromElement>SearchProfessors</awo:fromElement>
27      <awo:fromAttribute>section</awo:fromAttribute>
28      <awo:mapsTo rdf:resource="&cwo;CheckBox"/>
29  </awo:MultipleChoices>
30
31  <awo:MultipleChoices rdf:ID="SearchStudents">
32      <awo:fromElement>SearchProfessors</awo:fromElement>
33       <awo:fromAttribute>section</awo:fromAttribute>
34      <awo:mapsTo rdf:resource="&cwo;CheckBox"/>
35  </awo:MultipleChoices>
36
37  <awo:MultipleChoices rdf:ID="SearchPapers">
38      <awo:fromElement>SearchProfessors</awo:fromElement>
39      <awo:fromAttribute>section</awo:fromAttribute>
40      <awo:mapsTo rdf:resource="&cwo;CheckBox"/>
41  </awo:MultipleChoices>
42
43  <awo:IndefiniteVariable rdf:ID="SearchField">
44      <awo:mapsTo rdf:resource="&cwo;TextBox"/>
4546  </awo:IndefiniteVariable>
                                        ...
```

Figure 8. Example of the OWL specification of the "Search" part of Figure 2.

The *CompositeInterfaceElement* element, in this case, has the properties: *fromIndex*, *isRepeated*, *mapsTo*, *abstractInterface*, and *hasInterfaceElement*. The *fromIndex* property in line 2 indicates to which navigational Index this element belongs. This property is mandatory if no antecessor element of type *compositInterfaceElement* has declared it. The association with the "idxSearch" navigation element in

line 2; enables the generation of the link to the actual code that will run the search. Even though this example shows an association with a navigation element, it could just as well be associated with a call to application functionality such as "buy."

The *isRepeated* property indicates if the components of this element are repetitions of a single type (false in this case). The *mapsTo* property indicates which concrete element corresponds to this abstract interface element. The *abtractInterface* property specifies the abstract interface that will be activated when this element is triggered. The *hasInterfaceElement* indicates which elements belong to this element.

The *ElementExhibitor* element has the *visualizationText* and *mapsTo* properties. The former represents the concrete object to be exhibited, in this case the string "Search."

The *MultipleChoices* element has the *fromElement*, *fromAttribute*, and *mapsTo* properties. The *fromElement* and *fromAttribute* property indicate the corresponding element and navigational attribute in the navigational ontology, respectively. The *IndefiniteVariable* element has the *mapsTo* property

4. The implementation architecture

Figure 9 outlines the implementation architecture. Starting with the SHDM navigation and abstract interface designs, the corresponding ontology instances are input into a JSP generator, which instantiates the interface as a JSP file using TagLibs. The interpreter uses the Jena library to manipulate the ontology information.

The actual TagLib code used is determined by the concrete widget definition that has been mapped onto the corresponding abstract widget. The abstract interface determines the nesting structure of elements in the resulting page. It is expected that the designer will group together functionally related elements.

In Figure 10 we illustrate parts of the JSP code that is generated of the "HomePage" abstract interface widget.

This code first establishes the URIs corresponding to the TagLibs. It is possible to use different instances of the TagLib implementations by changing this declaration. Thus, for each possible concrete widget, a different implementation of the TagLib code will generate the desired HTML (or any other language) rendition for that widget.

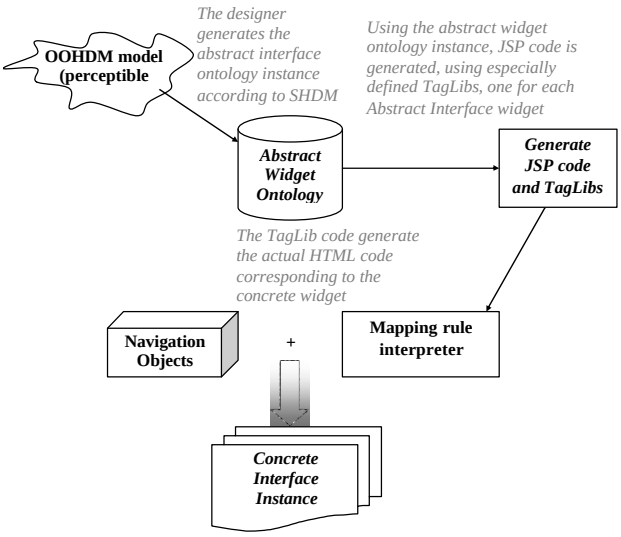

Figure 9. Outline of the implementation architecture

The actual values of navigation elements manipulated in the page are stored in Java Beans, which are declared first. The *element* property, generated in the JSP file, contains calls to the bean that the Tag Library uses to generate the visualized HTML code.

Our current simple implementation of the TagLib code simply wraps each element with a "DIV" CSS tag, with its own ID, and its CSS class defined according to its abstract widget type. In this way, we can attach CSS style sheets to the generated HTML to produce the final page rendering.

This concrete page definition format allows a large degree of flexibility for the graphic designer, given the expressive power of CSS, both in terms of layout itself and in terms of formatting aspects. Nevertheless, if a more elaborate page layout is desired, it is possible to edit the generated JSP page manually, altering the relative order of generated elements. For a more automated approach, it might be necessary to apply XSLT transformations to the JSP page.

```
<%@ page contentType="text/html; charset=ISO-8859-1" language="java" %>
<%@ page import="shdm.data.*" %>

<jsp:useBean id="idxMainMenu" class="shdm.data.Index" scope="request"/>
<jsp:useBean id="idxSearch" class="shdm.data.Index"  scope="request"/>
<%@ taglib uri='/WEB-INF/tlds/utilsTeste.tld' prefix="shdm"%>

<HTML> <HEAD> <TITLE>Concrete Home Page </TITLE> </HEAD> <BODY>
                ...
<DIV> <shdm:ElementExihibitor name="TitleSearch"
          mapsTo="Label" visualizationText="Search">
</shdm:ElementExihibitor> </DIV>

<DIV> <shdm:CompositeInterfaceElement name="Search"
          isRepeated="false"   mapsTo="Composition"
          fromIndex="idxSearch">

          <DIV> <shdm:CompositeInterfaceElement name="SearchElements"
                    mapsTo="Form"   isRepeated="false"
                    fromIndex="idxSearch"
                    abstractInterface="SearchResult">

                    <DIV> <shdm:MultipleChoices name="SearchProfessors"
                              element='<%=(Anchor)idxSearch
                                                .getEntry("SearchProfessors")
                                                .getAttribute("section")%>'
                              fromElement="SearchProfessors"
                              fromAttribute="section"
                              mapsTo="CheckBox">
                             </shdm:MultipleChoices> </DIV>

                    <DIV> <shdm:MultipleChoices  name="SearchStudents"
                              element='<%=(Anchor)idxSearch
                                                .getEntry("SearchStudents")
                                                .getAttribute("section")%>'
                              fromElement="SearchStudents"
                              mapsTo="CheckBox"
                              fromAttribute="section">
                    </shdm:MultipleChoices> </DIV>

                     <DIV> <shdm:MultipleChoices  name="SearchPapers"
                               element='<%=(Anchor)idxSearch
                                                .getEntry("SearchPapers")
                                                .getAttribute("section")%>'
                              mapsTo="CheckBox"
                             fromElement="SearchPaperrs"
                             fromAttribute="section">
                    </shdm:MultipleChoices> </DIV>

                    <DIV> <shdm:IndefiniteVariable  name="SearchField"
                               mapsTo="TextBox" cols="10">
                    </shdm:IndefiniteVariable> </DIV>

     </shdm:CompositeInterfaceElement> </DIV>
</shdm:CompositeInterfaceElement> </DIV>
                ...
</BODY> </HTML>
```

Figure 10. Generated JSP code.

5. Related work

Although we have not been able to identify other proposals exactly along the same lines as the one reported here, there are a few proposals, some commercial, aiming at generating "abstract interface" definitions similar to the ones proposed by SHDM, although typically at a lower level of abstraction – corresponding to the concrete widget ontology outlined here.

Among these proposals, we mention XUL [5], Laszlo System [6] User Interface Markup Language (UIML) [1] and the PIMA project at IBM [3].

The XUL language, based on XML, allows the definition of interfaces, which require an interpreter to be rendered. Similarly, the Laszlo system proposes an interesting interface architecture. An XML interface description is fed to an interpreter, written in Java, which in turn produces a Flash interface that is served to the client browser. This allows creating much richer interfaces than using HTML, even when using DHTML and JavaScript.

The UIML proposal is similar, but is able to describe the interface elements at a somewhat more abstract level, however still more concrete than the abstract interface proposed here. The implementation strategy for its various renderers is similar strategy than the one outlined here.

Our proposal could generate XUL, UIML or Laszlo code instead of HTML.

The PIMA project aims at defining a platform-independent application specification, and it aims at proposing similarly abstract descriptions of interface elements, among others.

The Web Modeling Language (WebML) [4] is a modeling approach for Web applications, similar to SHDM. In WebML, the closest counterpart to the abstract interface is the hypertext specification, but it is defined using elements characterized by more concrete functional properties. For example, a MultipleChoice abstract widget could correspond to both an IndexUnit and a MultiChoice IndexUnit. In other words, the components in a WebML page specification carry more application semantics than in our proposal.

6. Conclusions

We have presented the interface specification aspects of SHDM, and outlined the implementation strategy we are currently pursuing.

The ultimate evaluation of our approach is not so much the usability of the final interface as it is its capability of representing complex interaction patterns found in typical web applications. We have successfully described several such interfaces using the abstract widget ontology, and generated most of them. In some cases, complex designs require manual change in the generated nesting model.

We are investigating refinements to the abstract and concrete interface ontologies, to accommodate interfaces that are more complex. Another direction being pursued is examining the implementation architecture, both to try different template engines, such as Velocity, in the current version, and with other runtime environments, such as the ones mentioned in the previous section.

Finally, another line of investigation is exploring extensions to the model to accommodate adaptative hypermedia applications, as described in [2]. An important sub-problem here is the automatic generation of concrete interfaces, where usability guidelines, device

constraints, and user preferences are taken into account, at either "compile time," or during runtime.

Acknowledgement. This work was partially supported by grants from Conselho Nacional de Desenvolvimento Científico e Tecnológico – CNPq, Brazil, and from CAPES, Brazil.

7. References

[1] Abrams, M.; Helms, J. User Interface Markup Language (UIML) – Specification, 2002, available at http://www.uiml.org/specs docs/ uiml30-revised-02-12-02.pdf

[2] Assis, P.S.; Schwabe, D.; Barbosa, S.; "A Meta Model for Adaptive Hypermedia Applications", ED-Media 2004, Lugano, Switzerland, June 2004.

[3] Banavar, G.; Becky, J.; Gluzbergy, E.; Munson, J.; Sussman, J.; Zukowski, D. 2000, Challenges: An Application Model for Pervasive Computing, available at http://www.research.ibm.com/PIMA/Documents/Mobicom2000.pdf

[4] Ceri, S.; Fraternali, P.; Bongio, A. Web Modeling Language (WebML): a Modeling Language for Designing Web Sites, March 2000, available at: http://webml.org/webml/upload/ent5/1/ www9.pdf

[5] Deakin, N. Xul Tutorial, 2004, Disponível em: http://www.xulplanet.com/

[6] Laszlo System, Inc, Laszlo Tutorials 2002, Disponível em: http://www.laszlosystems.com/developers/learn/documentation/tutorials/index.php

[7] Schwabe, D.; Rossi, G. The Object-Oriented Hypermedia Design Model (OOHDM), 2003, Disponível em: http://www.telemidia.puc-rio.br/oohdm/oohdm.html

[8] Sun, http://java.sun.com/products/jfc/

[9] Assis, P. S.; Schwabe, D.; Barbosa, S.D.J., "Meta-models for Adaptive Hypermedia Applications and Meta-adaptation", Proc. of ED-Media 2004, forthcoming. Lugano, Switzerland, Jul. 2004.

[10] Corcho, O.; Gomez-Pérez, A.; López-Cima, A.; López-García, V., Suárez-Figueroa, M-d-C; "ODESeW. Automatic Generation of Knowledge Portals for Intranets and Extranets", Proceedings ISWC 2003, LNCS 2870, Springer Verlag, October 2003, pp 802 – 817. ISBN: 3-540-20362-1.

[11] Golbeck, J.; Alford, A. and Hendler, J.; Handbook *of Human Factors in Web Design*, chapter in Proctor, R; Vu, K-P. (eds) Organization and Structure of Information using Semantic Web Technologies. 2003 (available at http://www.mindswap.org/papers/Handbook.pdf).

[12] Jin, Y., Decker, S., Wiederhold, G.: "OntoWebber: Building Web Sites Using Semantic Web Technologies", http://www-db.stanford.edu/~yhjin/docs/owedbt.pdf.

[13] Lassila, O.; Swick, R.: "Resource Description Framework (RDF) Model and Syntax Specification", W3C Recommendation 22 February 1999, http://www.w3.org/TR/1999/REC-rdf--syntax-19990222/.

[14] Lima, F.: "Modeling applications for the Semantic Web", PhD Thesis, Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro, Brasil, 2003. (in Portuguese)

[15] Lima, F., Schwabe, D.: "Modeling Applications for the Semantic Web", In Proc. of the 3rd Int. Conference on Web Engineering (ICWE 2003), Oviedo, Spain, July 2003. Lecture Notes in Computer Science 2722, Springer Verlag, Heidelberg, 2003. pp 417-426. ISBN 3-540-40522-4.

[16] Lima, F.; Schwabe, D.: "Application Modeling for the Semantic Web", Proceedings of LA-Web 2003, Santiago, Chile, Nov. 2003. IEEE Press, pp. 93-102, ISBN (available at http://www.la-web.org).

[17] Rossi, G., Schwabe, D. and Lyardet, F.: "Web Application Models Are More than Conceptual Models" In Proc. of the ER'99, Paris, France, November 1999, Springer, 239-252.

[18] Schwabe, D. and Rossi, G.: "An object-oriented approach to Web-based application design" Theory and Practice of Object Systems (TAPOS), October 1998, 207-225.

[19] Smith, M.; McGuinness, D.; Volz, R.; Welty, C.: "Web Ontology Language (OWL) Guide Version 1.0", W3C Working Draft 4 November 2002,: http://www.w3.org/TR/owl-guide/

[20] van Harmelen, F.; Hendler, J.; Horrocks, I.; McGuinness, D.; Patel-Schneider, P.; Stein, L.: Web Ontology Language (OWL), Reference Version 1.0, W3C Working Draft 21 February 2003, http://www.w3.org/TR/owl-ref/