Design and Implementation of Semantic Web Applications

Daniel Schwabe, Guilherme Szundy, Sabrina Silva de Moura[1], Fernanda Lima[2]
[1]*Departamento de Informática, PUC-Rio*
{dschwabe, szundy, sabrina}@inf.puc-rio.br
[2]*Universidade Católica de Brasíia*
ferlima@ucb.br

Abstract

We present a software architecture to implement applications in the Semantic Web, based on designs specified in the Semantic Hypermedia Design Method (SHDM). This architecture supports the separation between Conceptual, Navigation and Interface models, and is based on direct manipulation of the various ontologies that define an SHDM design.

1 Introduction

In several recent proposals for RDF-based applications (e.g., [2][3][4]), there is a common (mostly unstated) underlying assumption that applications in the Semantic Web are, at least in many cases, browsers over RDF ontologies. In other words, most websites can be directly generated by mapping application domain ontologies into HTML interfaces.

A similar view was espoused by researchers in the Conceptual Modeling area, when first dealing with Web applications. We have argued elsewhere [9] that it is advantageous to separate conceptual modeling from navigation modeling, and from interface modeling. Each of these modeling activities address separate concerns in web applications that are best handled with specific modeling primitives.

In this paper, we present an approach for designing and implementing applications in the Semantic Web based on the Semantic Hypermedia Design Method [6][8]. Section 2 presents a summary of SHDM, with more emphasis on the Abstract Interface model; section 3 discusses an implementation architecture, and section 4 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. The artifacts produced by each phase are listed in Table 1.

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

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 [10] 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 is aimed at organizing the hyperspace, taking into account users' profiles and tasks.

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 [6].

The information items described in the Conceptual Model and in the Navigation Class Schema are resources specified in RDF [5]. The characterization of resources in SHDM is done using OWL[11][12], expressing

constraints (restrictions), enumeration and XML Schema datatypes.

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 updated only when new interface technologies are introduced.

2.1 Conceptual Model

The conceptual model is basically an object-style OWL model. In other words, it is an OWL model where some restrictions are followed, such as requiring that all properties have domain and range defined.

In Figure 1, we show an example of a small conceptual model for an academic department. Appendix 1 shows part of the equivalent OWL ontology which corresponds to an example of artifact 1.

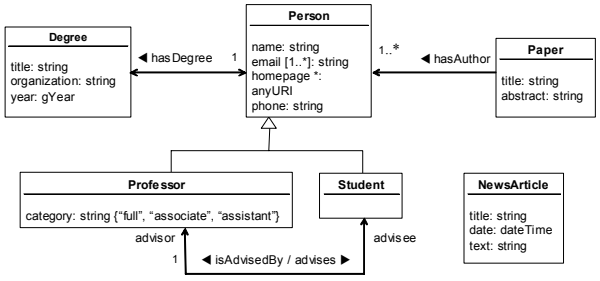

Figure 1. A simple conceptual model for an academic department

2.2 Navigational Model

An important tenet of OOHDM, followed by SHDM, is the realization that navigation objects are actually *views* over conceptual objects [9]. The SHDM Navigational Design defines a navigational vision of the Conceptual Design, specifying the information that will be processed, and the possible navigations among them, according to user profiles and tasks to be supported.. During the navigational design we are interested in specifying:

- which objects can be reached by the user (the navigational nodes);
- which relations exist among these navigational nodes (the links);
- within which sets of objects the user will navigate (the contexts);
- in which ways these sets will be accessed (the access structures);
- which different content must be presented to the user, depending on the context he is in (the inContext classes).

In Figure 2 we show an example of a navigational class model based on the conceptual model defined in Figure 1.

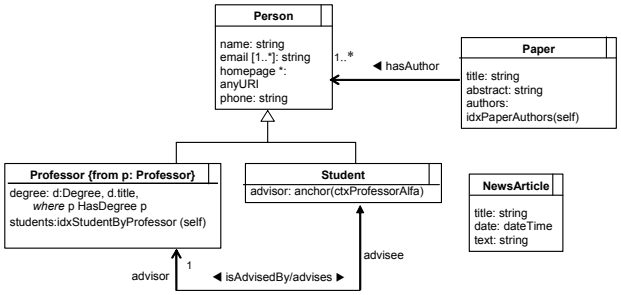

Figure 2. A navigational model based on the conceptual model in Figure 1.

The mapping between the conceptual ontology and the navigational ontology can be seen in this example by observing the attribute "degree" defined for navigational class "Professor", based on the conceptual class with the same name. It is assumed that attributes with the same name as conceptual attributes are simply copied to the navigational model.

Certain navigational class attributes also employ navigational ontology primitives such as idx, which stands for an index (a collection of references). An example is attribute "authors" for navigational class "Paper", which is an index to its authors. Similarly, attribute "advisor" for class "Student" is an anchor to an instance of "Professor", in the context "ProfessorByStudent".

The navigational space is defined in SHDM using the notion of contexts, which are sets of meaningful (for the task) navigational objects. Elements of a context are defined through a query. For example, the context "Professor Alpha" contains all Professors, ordered alphabetically; "Student ByProfessor" contains all the

students "AdvisedBy" each professor, which is, in fact, a set of contexts, one for each professor.

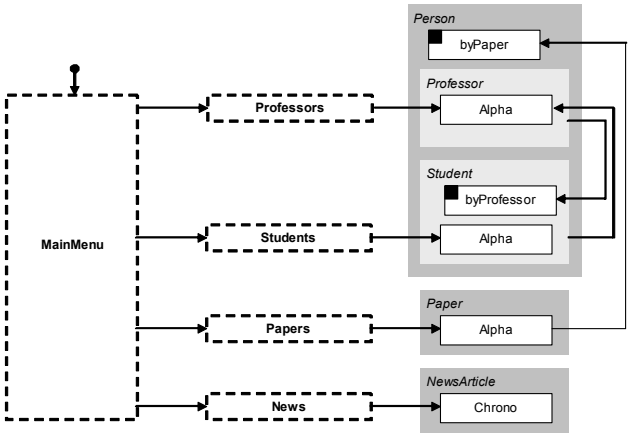

Figure 3. An example of context diagram for an academic department website.

The full definition of a context is given in its Context Definition Card, exemplified below. Notice the query expression defining the members of the Student ByProfessor context.

Context: Professor Alpha
Parameters:
Elements: prof Professor
Context class:
Order: prof.name ASC
Navigation: index (idxProfessorsAlfa), sequential
Operations:
Restrictions:
Comments: All professors in alphabetical order of name.

Context: Student ByProfessor
Parameters: prof Professor
Elements: stdt Student WHERE stdt isAdvisedBy prof
Context class:
Order: stdt.name ASC
Navigation: sequential
Operations:
Restrictions:
Comments: All students advised by a given professor.

There are analogous cards to define access structures, as exemplified below for the index of "Person ByPaper".

Index: idxPaperAuthors	
Parameters: pp Paper	
Elements: pers Person WHERE pp has Author pers	
Attributes: pers.name	**Target:** PersonByPaper(pers, pp)
Order: prof.name ASC	
Restrictions	
Comments: Index listing all the authors of a paper.	

2.3 Abstract Interface Model

Whereas Navigation design focuses on supporting users in achieving their intended tasks, 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 really just another (albeit distinguished) application functionality.

Since this information exchange is driven by the tasks being supported, 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 more concrete level, it is necessary to define the actual look and feel of the application, including layout, font, color and graphical appearance. This is typically carried out by graphics designers. 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.

The most abstract level is called the Abstract Interface, focuses 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 4.

An abstract interface widget can be any of the following:

- EventActivator, which is capable of reacting to external events;
- ElementExihibitor, which is able to exihibit some type of content;
- 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...;
- A composition of any of the above.

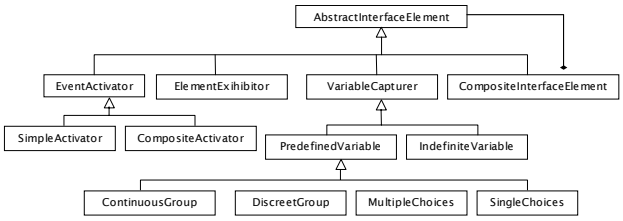

Figure 4. Abstract Widget Ontology

Any interface can be described as a composition of abstract interface widgets. In Figure 5, we show an example of an interface, and part of its corresponding abstract interface ontology.

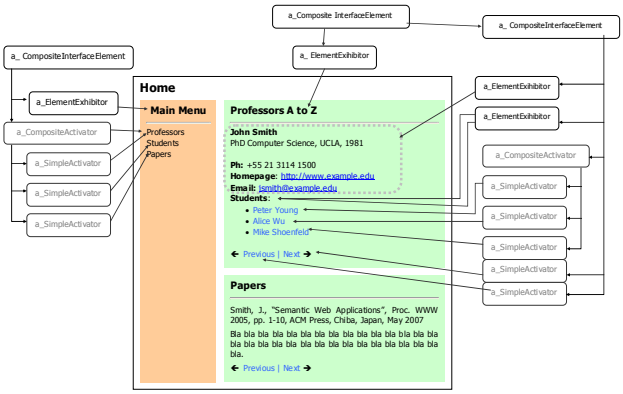

Figure 5. An example of an Abstract Interface and corresponding Abstract Widgets

Abstract interface widgets must be mapped onto concrete interface widgets in order to be perceived in the actual interface. Concrete widgets are specified in another simple ontology, shown in Figure 6.

```
<ConcreteInterfaceElem rdf:ID="VertScrollBar"/>
<ConcreteInterfaceElem rdf:ID="Image"/>
<ConcreteInterfaceElem rdf:ID="Form"/>
<ConcreteInterfaceElem rdf:ID="HorizScrollBar"/>
<ConcreteInterfaceElem rdf:ID="RadioButon"/>
<ConcreteInterfaceElem rdf:ID="ComboBox"/>
<ConcreteInterfaceElem rdf:ID="CheckBox"/>
<ConcreteInterfaceElem rdf:ID="TextBox_MultiLine"/>
<ConcreteInterfaceElem rdf:ID="Text"/>
<ConcreteInterfaceElem rdf:ID="Label"/>
<ConcreteInterfaceElem rdf:ID="Link"/>
<ConcreteInterfaceElem rdf:ID="Button"/>
<ConcreteInterfaceElem rdf:ID="TextBox_SingleLine"/>
```

Figure 6. Concrete Interface Widgets Ontology

The actual mapping is part of the Abstract Interface Ontology, as illustrated in Figure 7. In this snippet, it is stated that the "EventActivator" abstract interface widget can only be mapped into the "Link" or "Button" concrete interface widgets (see the grayed areas).

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 8 shows the specification of the "Previous Professor" (of class "EventActivator") abstract interface widget shown in Figure 5, which is mapped onto a "Link" concrete interface element.

```
<!DOCTYPE rdf:RDF [

        <!ENTITY awo "http://www.tecweb.inf.puc-rio.br/ontology/AW/awo#" >
        <!ENTITY cwo  "http://www.tecweb.inf.puc-rio.br/ontology/CW/cwo# " >
<rdf:RDF
        xmlns:awo  = "&awo;"
        xmlns:cwo = "&cwo;"> ]>
        ...
<owl:Class rdf:ID="EventActivator">
   <rdfs:subClassOf rdf:resource="#AbstractInterfaceElement" />
   <rdfs:subClassOf>
      <owl:Restriction>
         <owl:onProperty  rdf:resource="&awo;mapsTo" />
        <owl:allValuesFrom>
          <owl:Class>
            <owl:oneOf rdf:parseType="Collection">
               <wc: ConcreteInterfaceElem rdf:about="&cwo;Link" />
               <wc: ConcreteInterfaceElem rdf:about="&cwo;Button" />
            </owl:oneOf>
              </owl:Class>
          </owl:allValuesFrom>
      </owl:Restriction>
    </rdfs:subClassOf>
  </owl:Class>
... </rdf:RDF>
```

Figure 7. Mapping specification between Abstract and Concrete Interface widgets.

```
<!DOCTYPE rdf:RDF [

        <!ENTITY awo "http://www.tecweb.inf.puc-rio.br/ontology/AW/aws#" >
        <!ENTITY cwo  http://www.tecweb.inf.puc-rio.br/ontology/CW/wc# " >
<rdf:RDF
        xmlns:awo  = "&awo;"
        xmlns:cwo = "&cwo;">
...
<awo:EventActivator rdf:ID="ProfessorPrevious">
<awo:mapsTo rdf:resource= "&cwo;Link" />
<awo:NavigationElement>
        <-- name of the navigational element that is represented by this
         abstract element -->
</awo:NavigationElement>
  </awo:EventActivator>
```

Figure 8. Mapping between abstract interface widget and navigation element.

The mapping onto the concrete widget ontology will record the actual interface elements chosen by the designer. The nesting structure of the abstract interface must be mapped onto the actual layout. At the moment, we assume that layout information will be specified using the CSS box model, since each abstract element will correspond to a box (DIV tag in HTML). The desired layout is specified in a style sheet attached to the final rendered page.

In the next session, we outline the implementation architecture.

3 Implementation Architecture

We describe the implementation architecture by showing the flow of processing a request that is issued by a client (usually, a browser). Figure 9 shows the main components; the sequence of events is keyed to the numbers in the diagram.

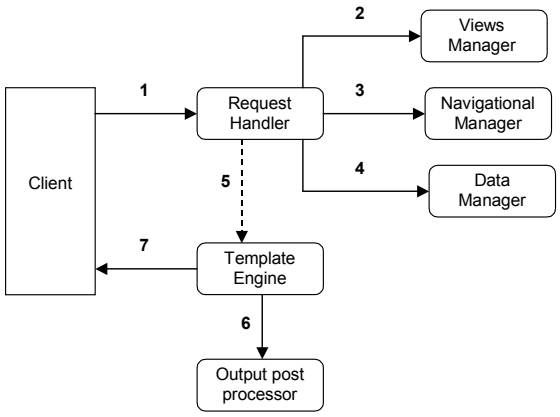

Figure 9. Main components of the implementation architecture, and flow of events.

Before describing some of the details, it should be mentioned that each abstract interface is internally represented by a *view*, which contains a summary of all navigational elements contained in that abstract interface. By examining a view, it is possible to quickly determine all navigation elements that must be retrieved or computed.

The handling of a request works as follows:

1. The *Client* sends a request to the system. The request is received by the *Request Handler* and must contain the name of the view to be presented and any parameters required by the navigational structures that compose this view.
2. The Request Handler consults the Views Manager to obtain a view definition for the view name received in the request. This view definition is obtained from the Abstract Interface definition.
3. From the view definition the Request Handler obtains a list with the name of all navigational structures that compose the view. For each name on the list it will consult the Navigational Manager to obtain a navigational structure definition, obtained from the navigational space specification. For each navigational structure definition it is verified if all the parameters required by the structure where provided in the request.
4. The Request Handler uses the Data Manager to retrieve the data for each navigational structure presented in the view, given its definition and the parameters passed in the request. The data is returned encapsulated in objects that represent the navigation components (nodes, indexes, contexts)
5. All the data retrieved is passed to Template Engine, which takes over the processing from the Request Handler. It fills the template for the view to be shown, with all the retrieved data, producing the output The template itself was previously created based on the Abstract Interface definition and the Concrete Widgets Ontology.
6. Optionally the output can be post-processed to perform additional format transformations (e.g. the output can be XML that is post processed to be converted to HTML by applying an XSLT transformation)
7. The output in its final format is returned to the client.

Our current implementation uses JSP tag libraries to represent abstract interface widgets. These tag libraries point to code that implements the mapping for each abstract interface widget. This code looks up the mapping ontologies, and runtime configuration parameters. This allows generating different interface code depending on a variety of parameters, such as the device being used by the user – it can produce plain HTML if the user is using a computer, or WML if the user is using a cellular telephone.

The actual RDF data is kept in an RDF store accessed using the JENA library. We are also experimenting with Sesame for this function.

4 Conclusions and Future Work

We have outlined in this paper how an SHDM design, specified through several ontologies, can be directly mapped onto executing code. A first prototype of the running system is being finalized.

There are several short and long term research and development aspects that we will be pursuing further.

From a methodological point of view, we intend to validade the various models, notably the Abstract Interface, against industrial grade applications.

There are several primitives in SHDM that still need refinement, such as faceted navigation and anonymous classes.

Given the ability to handle both schema-level and instance-level data, it is a natural extension to SHDM to be able to handle adaptive applications. In particular, we are looking into supporting meta-adaptation , where the type of adaptation itself varies depending on various parameters (see [1]).

From an implementation point of view, we are investigating various alternatives with respect to implementation environments, including alternative

persistence mechanisms, application servers, and interface technologies.

We also plan to study the scalability of our approach.

5 References

[1] 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.

[2] 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.

[3] 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).

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

[5] 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/.

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

[7] 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.

[8] 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).

[9] 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.

[10] 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.

[11] 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/

[12] 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/

Appendix I – An extract of the OWL specification for the conceptual model for an academic department

```
<?xml version="1.0"?>

<!DOCTYPE rdf:RDF [
        <!ENTITY cOnt
"http://www.tecweb.inf.puc-rio.br/shdm/example/cOnt.owl.xml#" >
        <!ENTITY shdm "http://www.tecweb.inf.puc-rio.br/shdm.owl.xml#" >
        <!ENTITY owl  "http://www.w3.org/2002/07/owl#" >
        <!ENTITY rdf  "http://www.w3.org/1999/02/22-rdf-syntax-ns#" >
        <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#" >
        <!ENTITY xsd  "http://www.w3.org/2001/XMLSchema#" >
]>

<rdf:RDF
        xmlns     = "&cOnt;"
        xml:base  = "&cOnt;"
        xmlns:shdm = "&shdm;"
        xmlns:owl  = "&owl;"
        xmlns:rdf  = "&rdf;"
        xmlns:rdfs = "&rdfs;"
        xmlns:xsd  = "&xsd;">

        <owl:Ontology
rdf:about="http://200.165.173.38:8080/shdm/example/cOnt.owl.xml">
                <owl:imports
rdf:resource="http://200.165.173.38:8080/shdm/example/shdm.owl.xml"/
>
        </owl:Ontology>

<owl:Class rdf:ID="Person">
        <rdfs:label>Person</rdfs:label>
        <!-- A Person has one and only one name -->
        <rdfs:subClassOf>
                <owl:Restriction>
                        <owl:onProperty rdf:resource="#personName" />
                        <owl:cardinality
rdf:datatype="&xsd;nonNegativeInteger">1</owl:cardinality>
                </owl:Restriction>
        </rdfs:subClassOf>
        <!-- A Person has at least one e-mail -->
        <rdfs:subClassOf>
                <owl:Restriction>
                        <owl:onProperty rdf:resource="#personEmail"/>
                        <owl:minCardinality
rdf:datatype="&xsd;nonNegativeInteger">1</owl:minCardinality>
                </owl:Restriction>
        </rdfs:subClassOf>
        <!-- A Person has at most one homepage -->
        <rdfs:subClassOf>
                <owl:Restriction>
                        <owl:onProperty rdf:resource="#personHomepage"/>
                        <owl:maxCardinality
rdf:datatype="&xsd;nonNegativeInteger">1</owl:maxCardinality>
                </owl:Restriction>
        </rdfs:subClassOf>
        <!-- A Person has one and only one phone number -->
        <rdfs:subClassOf>
                <owl:Restriction>
                        <owl:onProperty rdf:resource="#personPhone"/>
                        <owl:cardinality
rdf:datatype="&xsd;nonNegativeInteger">1</owl:cardinality>
                </owl:Restriction>
        </rdfs:subClassOf>
<!-- A Person has one and only one degree -->
        <rdfs:subClassOf>
                <owl:Restriction>
                        <owl:onProperty rdf:resource="#hasDegree"/>
                        <owl:cardinality
rdf:datatype="&xsd;nonNegativeInteger">1</owl:cardinality>
                </owl:Restriction>
        </rdfs:subClassOf>
</owl:Class>
...
```

```
<owl:DatatypeProperty rdf:ID="personName">
        <rdfs:label>name</rdfs:label>
        <rdfs:domain rdf:resource="#Person" />
        <rdfs:range rdf:resource="&xsd;string" />
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:ID="personEmail">
        <rdfs:label>email</rdfs:label>
        <rdfs:domain rdf:resource="#Person" />
        <rdfs:range rdf:resource="&xsd;string" />
</owl:DatatypeProperty>
...
<!-- ### Definition of class Professor ### -->
<owl:Class rdf:ID="Professor">
        <rdfs:label>Professor</rdfs:label>
        <rdfs:subClassOf rdf:resource="#Person"/>
        <!-- A Professor has one and only one category -->
        <rdfs:subClassOf>
                <owl:Restriction>
                        <owl:onProperty rdf:resource="#professorCategory"/>
                        <owl:cardinality
rdf:datatype="&xsd;nonNegativeInteger">1</owl:cardinality>
                </owl:Restriction>
        </rdfs:subClassOf>
</owl:Class>

...

<!-- ### Relationship defintions ### -->

<!-- Person hasDegree Degree -->
<owl:ObjectProperty rdf:ID="hasDegree">
        <rdfs:label>hasDegree</rdfs:label>
        <rdfs:domain rdf:resource="#Person"/>
        <rdfs:range rdf:resource="#Degree"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="inverseOf_hasDegree">
        <owl:inverseOf rdf:resource="#hasDegree"/>
</owl:ObjectProperty>


<!-- Professor advises Student  / Student isAdvisedBy Professor -->
<owl:ObjectProperty rdf:ID="isAdvisedBy">
        <rdfs:label>isAdvisedBy</rdfs:label>
        <rdfs:domain rdf:resource="#Student"/>
        <rdfs:range rdf:resource="#Professor"/>
        <shdm:domainRole rdf:datatype="&xsd;string">advisee</shdm:domainRole>
        <shdm:rangeRole rdf:datatype="&xsd;string">advisor</shdm:rangeRole>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="advises">
        <rdfs:label>advises</rdfs:label>
        <owl:inverseOf rdf:resource="#hasDegree"/>
</owl:ObjectProperty>

<!-- Paper hasAuthor Person -->
<owl:ObjectProperty rdf:ID="hasAuthor">
        <rdfs:label>hasAuthor</rdfs:label>
        <rdfs:domain rdf:resource="#Paper"/>
        <rdfs:range rdf:resource="#Person"/>
</owl:ObjectProperty>

</rdf:RDF>
```