Design and Implementation of Linked Data Applications Using SHDM and Synth

Mauricio Henrique de Souza Bomfim and Daniel Schwabe

Informatics Department, PUC-Rio Rua Marques de Sao Vicente, 225.
Rio de Janeiro, RJ 22453-900, Brazil
mauriciobomfim@gmail.com, dschwabe@inf.puc-rio.br

Abstract. In this paper, show how Linked Data Applications (LDAs) can be designed and implemented using an evolution of the Semantic Hypermedia Design Method, SHDM, and a new development environment supporting it, Synth. Using them, it is possible to take any RDF data available on the Linked Data cloud, extend it with one's own data, and provide a Web application that exposes and manipulates this data to perform a given set of tasks, including not only navigation, but also general business logic. In most cases, the only code that needs to be written is for the Business Logic; the remainder code is automatically generated by Synth based on the SHDM models.

Keywords: Linked Data, Semantic Web, Linked Data Applications, RDF, Design Method, Model Driven Development.

1 Introduction

Up until recently, Web applications followed, to a great extent, the traditional data intensive development approaches. The prevailing paradigm is one where organizations or individuals are responsible for either creating or collecting, through the application itself, all its data, which remains under its management. With the advent of the Semantic Web, and especially the more recent growth of the Linked Data initiative [2], we are witnessing the emergence of the Linked Open Data (LOD)[1] cloud, a collection of interlinked data sources spanning a wide range of subjects. It is now quite feasible to design Web applications (hosted in websites) whose contents are at least partially drawn from the LOD cloud. For example, several sites at the BBC (e.g. 2010 World Cup website[2]) routinely make use of data pulled from the LOD cloud.

This new scenario has given rise to a new challenge – how to effectively build applications that consume linked data, often combining with locally generated and managed data – that we will call "Linked Data Applications", LDAs in short. At first sight, this may sound similar to the problem of building traditional Web applications – after all, the Semantic Web is part of the traditional, so-called "Web of documents". For such Web applications a number of design methods have been proposed, e.g., OOHDM [9], SHDM [4], WebML [2], UWE [3], Hera [13], among others.

But LDAs present additional challenges. First, one of its basic tenets is the reuse of existing information – both vocabularies and instance data, whenever possible.

[1] http://linkeddata.org
[2] http://www.bbc.co.uk/blogs/bbcinternet/2010/07/bbc_world_ cup_2010_dynamic_sem.html

S. Auer, O. Diaz, and G.A. Papadopoulos (Eds.): ICWE 2011, LNCS 6757, pp. 121–136, 2011.
© Springer-Verlag Berlin Heidelberg 2011

Second is the principle that the data should carry as much machine-processable semantics as possible. Following this principle, it should be expected that, within feasible limits, the application semantics also be captured in machine processable way.

From the model-driven design (MDD) [11] point of view, this is not a new idea. Software development, according to MDD, is a process whereby a high-level conceptual model is successively translated into increasingly more detailed models, in such a way that eventually one of the models can be directly executed by some platform. To be consistent with the LDA philosophy then, they should be specified using models expressed in the same formalisms used to describe the data itself.

There are some proposals of development environments or frameworks for supporting the development of LDAs, such as CubicWeb[3], the LOD2 Stack[4], and the Open Semantic Framework[5]. In addition, semantic wiki-based environment such as Ontowiki[6], Kiwi[7], and Semantic Media Wiki[8] have also been used as platforms for application development over Linked Data.

While useful, they do not present a set of integrated models that allow the specification of an LDA, and the synthesis of its running code from these models. Therefore, much of the application semantics, in its various aspects, remains represented only in the running implementation code.

On the other hand, existing development methodologies, such as OOHDM, WebML, UWE, and Hera, do not have the primitives to directly implement LDAs, although some are able to import and manipulate RDF data.

In this paper we present an updated version of SHDM [7], the Semantic Hypermedia Design Method, and Synth[9], a development environment that allows the specification of LDAs according to SHDM models, and the generation of running code from this specification. Using Synth, it is possible to generate LDAs with little or no programming, simply by declaring models as proposed by SHDM.

The remainder of this paper is organized as follows. Section 1 presents a motivating example that will be used throughout the paper to illustrate the various issues being discussed; this application is built over data published in the Semantic Web, enriching it with one's own private data. Section 2 presents a brief summary of SHDM, focusing on its proposed models and discussing their relevance for LDAs. Section 3 presents Synth, the development environment supporting SHDM, and shows how the example application has been implemented. Finally, Section 4 discusses related and future work.

2 A Working Example

In order to facilitate the presentation and discussion of our approach, we first present an example LDA that illustrates a typical scenario. We will use it as a

[3] http://www.cubicweb.org
[4] http://lod2.eu/WikiArticle/TechnologyStack.html
[5] http://openstructs.org/open-semantic-framework
[6] http://ontowiki.net/Projects/OntoWiki
[7] http://www.kiwi-project.eu
[8] http://www.semantic-mediawiki.org/wiki/Semantic_MediaWiki
[9] http://www.tecweb.inf.puc-rio.br/synth

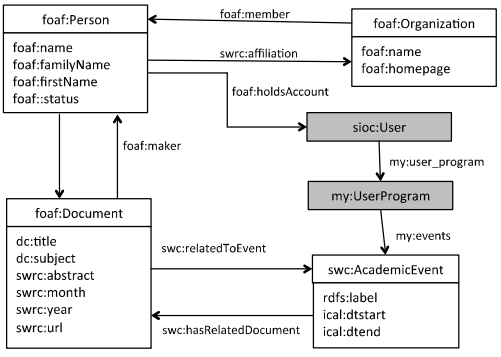

Fig. 1. Domain model for a Personalized Conference Schedule application

reference to exemplify certain aspects of the proposed approach in the remainder of the paper.

Many conferences make the metadata about its events (keynote talks, technical and scientific paper presentations, tutorials, etc.) publicly available as RDF data, such as the Semantic Web Conference series (see data.semanticweb.org). As is often the case, a conference attendee must choose which events she is interested in, and assemble a personal schedule of events to attend. We show a simple application that allows the attendee to use the available conference metadata published in the LOD to build a personal schedule.

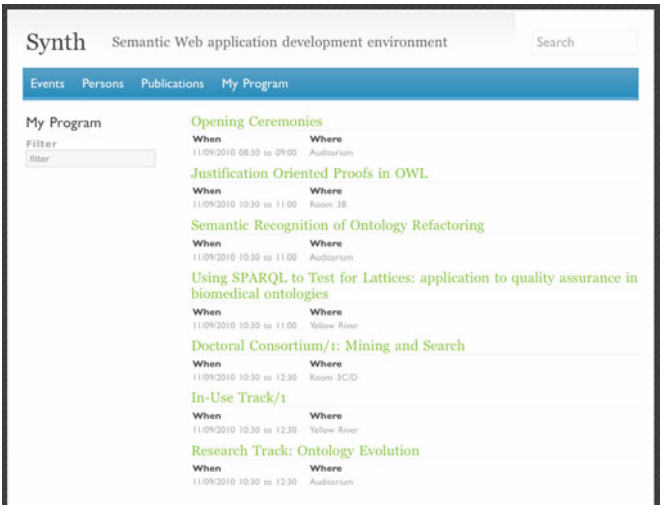

Fig. 2. A list of events at a conference, with the option to select for inclusion in personal schedule

Figure 1 shows the Domain model of the example application, using an UML-like notation. Classes in white occur in the input data taken from data.semanticweb.org (only a small fragment of the entire vocabulary was needed, depicted in this schema). The classes in gray were added to model the semantics of the example application.

The *sioc:User* class models the user of the application, and the *my:UserProgram* class models the personalized program, which will be a set of *swc:AcademicEvents*.

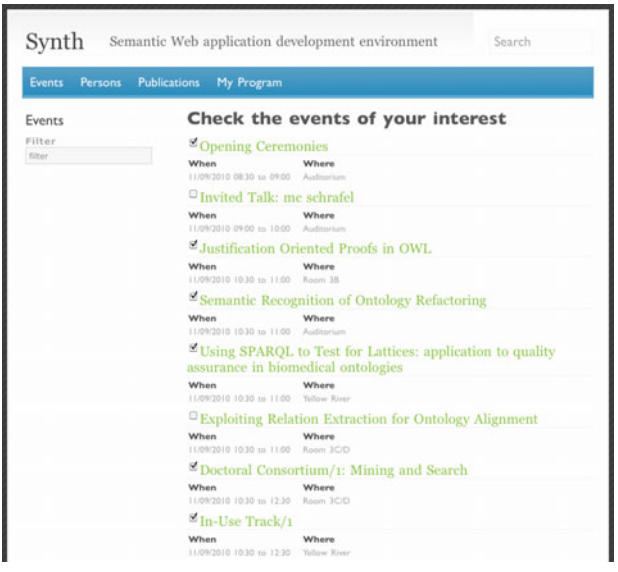

Fig. 3. A personal schedule of events

The starting point in the application is an interface that lists all events in chronological order, as illustrated in Figure 2. For each event, the user has the option of checking a checkbox to indicate her interest.

Once the user has selected the events of interest, a personal program is shown in Figure 3

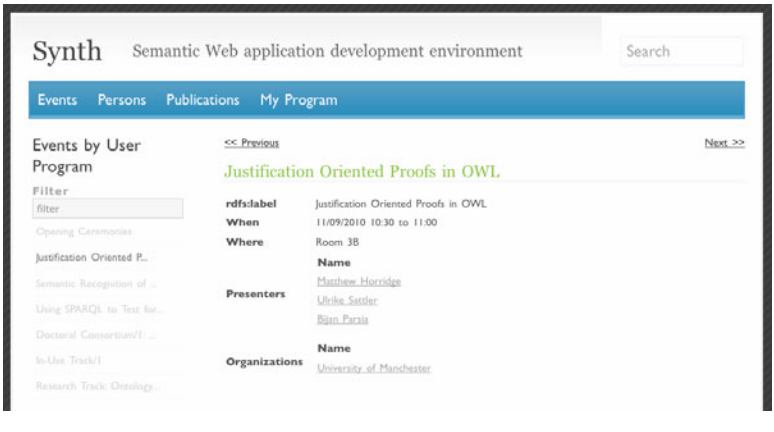

Fig. 4. Details of an event

In this interface, the user can click on an event name to see details, as illustrated in Figure 4. In this interface, clicking on a person's name will lead to an interface showing more data about that person. Clicking on an organization's name will show all events associated to that organization.
Notice that there is no direct relation in the Domain Model between *Organization* and *AcademicEvent*; rather, it must be computed as the *Organization* to which the *Authors* of the *Document* presented in the *AcademicEvent* belong.
We next discuss new or updated features of SHDM that deal with modeling LDAs.

3 The Evolution of SHDM

SHDM is a model-driven approach to design Semantic Web applications. Since it was first formulated, the Semantic Web itself has evolved, e.g., with the advent of the LOD cloud. Consequently, we have updated several aspects of SHDM, discussed in this section.

SHDM includes six different phases: Requirements Gathering, Domain modeling (formerly Conceptual Modeling), Hypertextual Navigational modeling (formerly Navigation Modeling), Abstract Interface modeling; Business Logic modeling (which did not exist previously), and Implementation. Each phase focuses on a particular aspect and produces artifacts detailing the application to be run on the web.

Before looking at SHDM details, we first discuss what MDD means in the context of Linked Data.

3.1 Linked Data, Applications and MDD

Linked Data, by itself, focuses only on providing information encoded as RDF, possibly following an RDFS or OWL schema, in such a way that it is possible for a program to process each information item, and to obtain additional related information when available. Thus, it provides but a small portion of the necessary specifications needed build a full-fledged application.

The only semantics of an RDF graph, is given by the RDF (and RDFS) metamodel semantics as specified in the standard[10]. This captures a very limited fraction of the "broad meaning" of the statements in the graph. The semantics restrict the possible interpretations of the set of triples as referring to some "real world" entity or objects, and their relations. The programs that manipulate these statements add additional specific semantics, as their behavior is determined in some way by particular sets of triples they receive as inputs.

In MDD, a set of models determines the ultimate application behavior. Each model captures some aspect of the application – e.g., data (domain), interface, hypertextual navigation, etc. If these models are specified as RDF statements using some vocabularies, specific interpreters that implement the desired behavior must provide the intended semantics. In fact, for RDFS and OWL, inference engines play this role with respect to the data semantics, but there is no counterpart for application behavior (i.e., the business logic).

[10] http://www.w3.org/TR/2004/REC-rdf-mt-20040210/

In SHDM, each of the proposed models establishes a specific vocabulary, and the semantics are currently given through the code generated using the Synth environment, presented later in this paper.

We will next describe the more relevant models in SHDM, as they apply to LDAs, pointing out evolutions with respect to the originally proposed method.

3.2 Domain Modeling

The Domain model characterizes the universe of discourse of a particular application. In most proposed methodologies, the (equivalent to the) Domain model is specified through a Class Schema (in OO methods) or ER schema (in database oriented methods). In contrast, in SHDM a Domain model is simply a set of RDF triples, which form a graph, which may include RDFS or OWL definitions. In the extreme case, there may not exist any Class definitions in Domain Model, just instances of resources representing information items.

This is an evolution over earlier versions, where the domain model was specified in SHDM's own vocabulary. Therefore, SHDM can use any Conceptual Modeling technique that is capable of producing RDF graphs, or none at all, if the starting point is an already existing RDF graph.

However, since we are focusing on LDAs, it is necessary to add an additional abstraction to SHDM's metamodel, to capture the concept of "datasets". Datasets are described using the VoID vocabulary[11], which was designed precisely for this purpose. Using datasets, it is possible to use RDF graphs stored in the LOD cloud as part of an application's Domain Model. We will later show how this is achieved.

3.3 Business Logic Modeling

LDAs – indeed, all applications - are built to perform computations over data instances of the Domain Model, causing effects that change the data and present some results to the user. This computation is usually defined to support a set of tasks and users, the intended audience of the application, which sometimes can be quite broad. The Business Logic of the application specifies these computations.

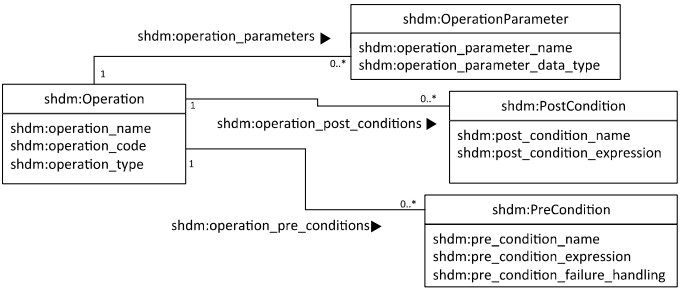

Fig. 5. The Operation metamodel in SHDM

[11] http://vocab.deri.ie/void/guide

In SHDM, he Business Logic model is given by specifying a set of *Operations* that will be made available by the application, much in the same way as services in service-oriented applications. SHDM's metamodel for Operations is shown in Figure 5. It is similar to Web Services specifications in other vocabularies (e.g., OWL-S[12], WSML[13]). with a few additions, described next.

Operations have properties specifying *Parameters*, and *Pre-* and *Post-conditions*. The *shdm:operation_name* property provides a label for the operation. The *shdm:operation_type* can be either "Internal" or "External". The latter are operations that can be invoked from outside the application, i.e., they are visible on the Web and may be invoked as a REST service. The former are operations that can only be accessed from within the application and are not visible outside..

The actual behavior specification of the operation is given in the *shdm:operation_code* property. The SHDM metamodel is agnostic as to how this code is given, so the value of this property is currently specified as a string. In principle, this string can be code in any language for which there is an interpreter. This can be a general programming language, or a Domain Specific Language [12], such as BPEL[14] or BPMN[15]. Strictly speaking, the value of this property could also be another RDF graph representing a particular DSL (e.g. BPMN), but we have not explored this. It should also be noted that Operations may be called from within other Operations.

The effective implementation behavior of the application is achieved by integrating the code DSL interpreter with the runtime engine – i.e., it must be able to access and change the values of the data.

3.4 Hypertextual Navigation Modeling

The term "Navigation Design" or "Hypertext Design" has been used to designate the activity of designing the "navigation topology" of Web applications [9], which also applies to LDAs. The main idea is that applications aim at supporting a given set of tasks and, while performing a task, some of its steps are often best supported by providing a hypertextual navigation structure over the information items being processed.

For some domains, such as online newspapers and magazines, browsing and navigating is the main task to be performed, and hypertextual navigation is well suited to support it. For other domains, e.g. online stores, hypertextual navigation is useful to support only a few of the tasks, e.g., browsing the catalog section, but is of little help to support other tasks, e.g., the transactional (e.g., payment) processing.

Hypertextual navigation modeling prescribes preferred navigation paths through the information items to support a given set of tasks. When specifying the navigation options, it is useful to be able to refer to sets of items that share similar navigation alternatives, instead of individual ones. For example, it is better to state that "when accessing any *Person*, one may navigate to one of the *Papers* s/he has created, and from any *Paper*, continue navigating to other *Paper*s created by that same *Person*", instead of specifying this linking structure repeatedly for every instance of *Person* and *Paper*. Notice that several hypertextual links are induced by the task, and are not present in the Domain Model – e.g., "next" and "previous" *Paper* created by a *Person*.

[12] http://www.w3.org/Submission/OWL-S/
[13] http://www.wsmo.org
[14] http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html
[15] http://www.bpmn.org

Following OOHDM, the major primitive for specifying the hypertextual navigation topology of LDAs is the *Context*. A *Context* is a set of resources that share similar hypertextual alternatives, e.g. “Papers by a Person”, “Products in a Department”, and “Stories in a Section”. In a way, a Context plays the analogous role with respect to hypertextual navigation as Classes (in OO languages) play for structure and behavior. The same way the Class definition determines the structure and behavior of its instances, the Context definition determines the navigation alternatives of its elements.

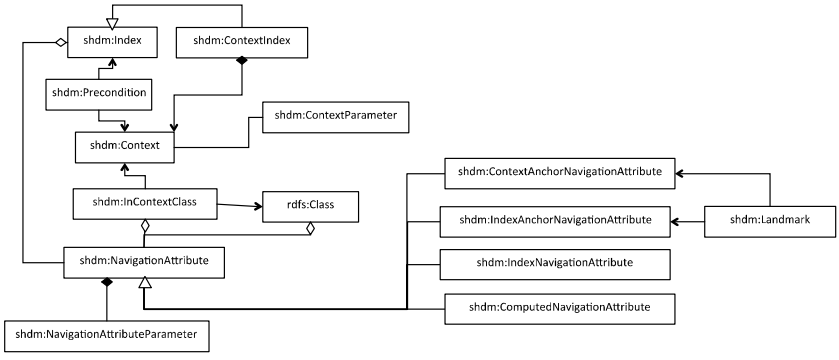

Fig. 6. SHDM Hypertextual Navigation metamodel

RDF data constitutes a graph, which can be considered as a “node-and-link” hypertext. For LDAs, this “node-and-link” model is often too low level, not supporting the set-based specification of contexts. In other words, it is necessary to extend the pure RDF graph of the Domain Model with additional properties that will drive the higher-level Hypertext navigation.

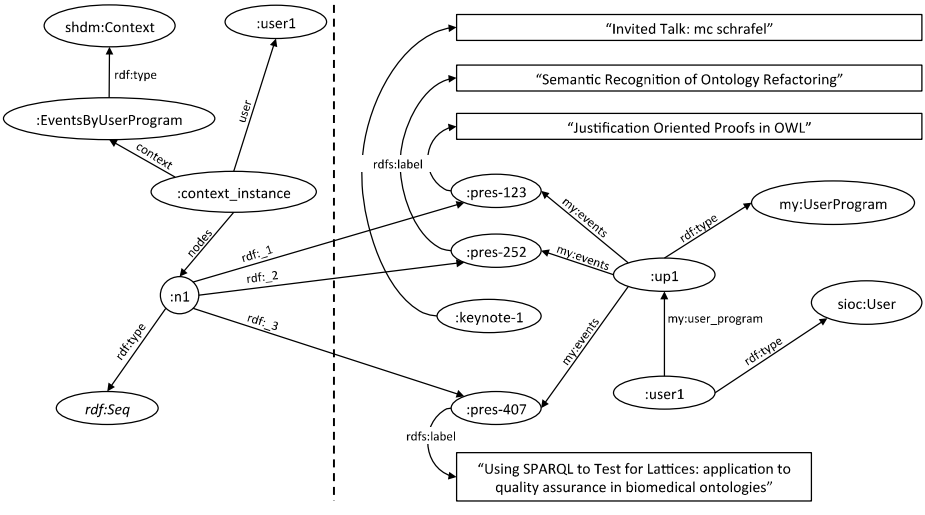

Fig. 7. A fragment of the hypertextual navigation graph (left side) for the example

Accordingly, SHDM's navigation model is defined as a set of additional properties assigned to RDF resources in the Domain model. The hypertextual navigation behavior defined by this model is implemented as part of the runtime environment of Synth. Figure 6 shows the Hypertextual Navigation metamodel for SHDM.

A full description of SHDM's Hypertextual Navigation metamodel is outside the scope of this paper; more details can be found in [4]. Nevertheless, we illustrate how the Domain model is extended with navigation properties at runtime, to allow the interpreter to implement the hypertextual navigation behavior specified in the corresponding model.

Figure 7 shows a fragment of the data graph of our example application on the right hand side, and the hypertextual navigation runtime structure on the left hand side. This fragment shows the navigation structure of the instance of the "EventsByUserProgram" context (a "context_instance" node) for "user1", which is the value of the "user" parameter of the context. In this case, "pres-12", "pres-252" and "pres-407" are part of this context. Notice that the "context_instance" node is used to group the context elements and to provide ordering among them.

A more careful analysis of what actually *is* hypertextual navigation reveals that it is ultimately just one (more) kind of application behavior, among all behaviors defined by the Business Logic. However, differently from any arbitrary behavior one might have in an application, which usually is domain-dependent, the hypertextual navigation behavior has a well-known semantics, which should also be modeled as an operation in the Business Logic model. In other words, what navigation means is known *independently* of any particular domain, in advance.

From this observation, we define the hypertextual navigation in the development support environment simply as a *pre-defined* set of operations. In other words, navigation operations are operations whose code is predefined, and automatically integrated into the generated runtime environment.

There are other advantages in to modeling navigation as operations. For example, since operations have pre-conditions, it is possible to define conditional navigation behavior. For instance, in our example, if the user has not yet selected her preferred events, if she tries to navigate to the "My Program" index, the application will present a message and request the user to first select the preferred events.

3.5 Interface Modeling

The Interface Model in SHDM [15] is based on the idea that it is possible to separate the "essence" of the interface from its look-and-feel. This is achieved by decomposing the interface specification into an Abstract Interface Model, and a Concrete Interface Model.

In brief, the Abstract Interface focuses on the roles played by each interface widget in the information exchange between the application and the outside world, including the user. It is abstract in the sense that it does not capture the look and feel, or any information dependent on the runtime environment. The Concrete Interface model is responsible for the latter.

Summarizing the Abstract Interface meta-model, an (abstract) interface is a composition of abstract interface elements (widgets). These in turn can be an *ElementExhibitor,* which is able to show values; and *IndefiniteVariable*, which is able to capture

an arbitrary input string; a *DefinedVariable*, which is able to capture input values (one or several) from a known set of alternatives; and a *SimpleActivator*, which is able to react to an external event and signal it to the application.

From an *Abstract Interface*, a mapping specification made by the designer determines how each abstract widget will be mapped onto one or more *Concrete Interface* elements, and onto which *Operation*s. Notice that the mapping to *Operation*s unifies access to the Domain Model - if it is a CRUD operation of an element of the Domain; to the Navigation Model, if it is a Navigation operation; and to the general Business Logic, otherwise.

This model allows a cleaner separation between the Hypertext Navigation and Business models; there is no need to include both types of primitives in a single graph, as proposed in WebML or UWE. When an event (i.e., user click) is captured at the interface (via a *SimpleActivator* widget), the mapping to the operations (there can be more than one operation mapped to the same interface element) will determine which operations will be executed, regardless of their being navigation or not.

We have subsequently extended this model to allow specifying interface-only behavior, thus modeling modern Rich Application Interfaces [5], but we will not detail here for reasons of space.

We next present Synth, a development environment that supports the creation of LDAs using the SHDM method.

4 The Synth Development Environment

Synth is a development environment for building applications that are modeled according to SHDM. It provides a set of modules that receives, as input, models generated in each step of SHDM and produces, as output, the hypermedia application described by these models. Synth also provides an authoring environment that facilitates the adding and editing of these models through a GUI that can run on any web browser.

4.1 Software Architecture

The software architecture of Synth was designed to be independent of its implementation technology. It consists of a set of modules, each responsible for maintaining and interpreting one of the models generated in each phase of SHDM. Each module is composed by a model described in a corresponding ontology in RDFS or OWL, and an interpreter that gives semantics to the models, in addition to the basic semantics of RDFS and OWL, in which they are represented. These modules work together, interpreting their models and communicating with each other, in order to generate the application runtime in accordance with the definitions of each model.

Figure 8 shows a conceptual view of the modular software architecture for applications modeled using SHDM. The gray boxes represent the modules and the white boxes represent the components of each module.

The persistence module handles the access and manipulation of application data. This module is composed of two layers: a storage, inferences and query layer and a RDF(S)/OWL mapping layer.

The storage, inferences and query layer provides a single interface for accessing multiple environments and platforms for RDF data. This layer converts the access interfaces of various RDF data environments (e.g., Jena[16], Sesame[17], Virtuoso[18] or OWLIM[19]) into a single interface known by the RDF(S)/OWL mapping layer, by applying the Adapter design pattern. This layer is also responsible for distributing queries among various data repositories, local or remote, combining their results, as enabled under the Linking Open Data initiative.

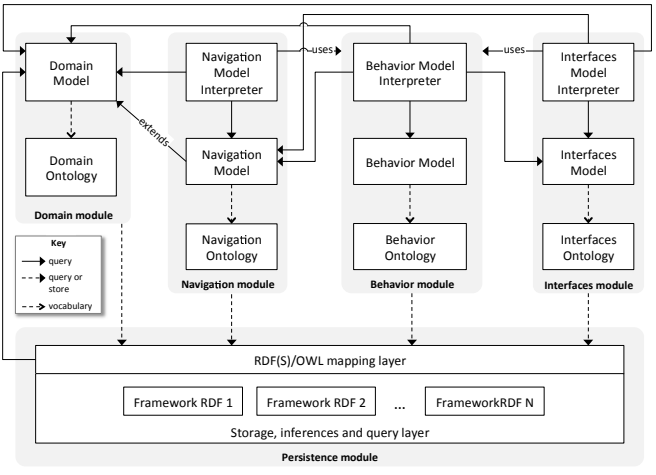

Fig. 8. Conceptual view of the Synth software architecture

The RDF(S)/OWL mapping layer provides a view of the data and meta data, originally persisted as RDF triples, as primitives of the programming language in which the application is implemented.

The other modules are the domain, navigation, business logic, and interfaces modules. Each of these maintains and interprets its corresponding models, and is similarly structured. The only exception is the domain module, which has no interpreter because the semantic of the domain model is that of RDFS and OWL, and the persistence module provides an interpreter for them in the RDF(S)/OWL mapping layer.

4.2 Module Collaboration

To illustrate how the modules collaborate, Figure 9 presents the sequence diagram showing the events in a typical execution of a "navigate" hypertextual navigation operation. The user interaction always happens through the external operations of the business logic model, available as Web Services. External agents invoke external operations by sending HTTP requests to the application. In this sense, external

[16] http://jena.sourceforge.net
[17] http://www.openrdf.org
[18] http://virtuoso.openlinksw.com/
[19] http://www.ontotext.com/owlim/index.html

operations fulfill the same role as the "controller" in the MVC-based (model-view-controller) architectures, coordinating user interactions, obtaining data from the domain Model, instructing the View to generate the interface and, finally, delivering the result to the user or external agent.

In this sequence diagram, the user sends a message to the business logic module invoking the method "execute" e informing the name of an operation, in this case "context", and some parameters for this operation. This operation performs the "contextual navigation" operation of the SHDM hypertextual navigation model.

After that, the external operation invokes the method "get_context" from the navigation module with the context identifier "AllPerson" as a parameter. The navigation model retrieves the "AllPerson" context definition, obtaining the query expression. This expression is then used to invoke the method "dsl" from the domain module, which simply passes it to the persistence module. This module converts this expression to an equivalent Federated SPARQL query expression, executes it getting the results as RDF triples, and maps these results into the programming language primitives.

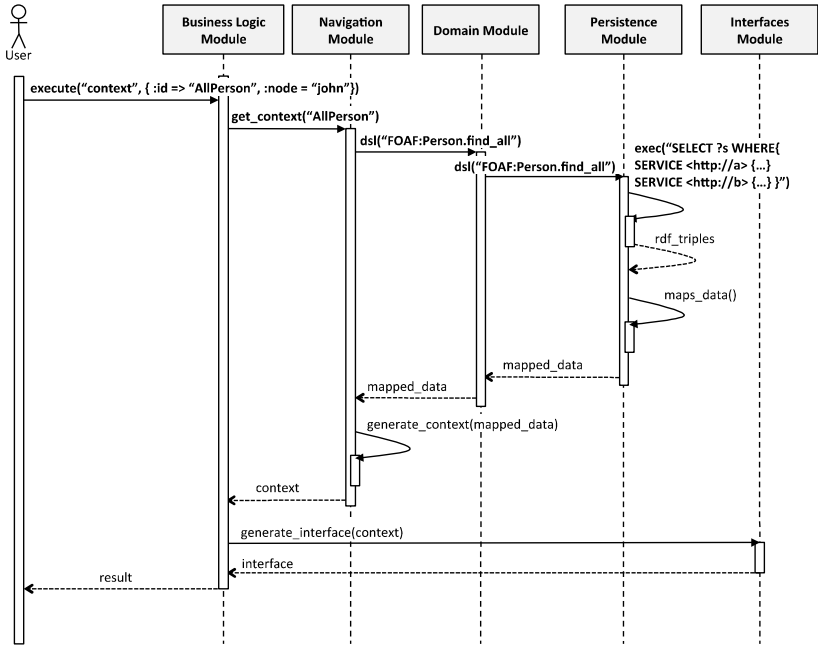

Fig. 9. Collaboration between modules in Synth

This mapped data is returned to the domain module and passed to the navigation module, which generates the context instance with its nodes based on the received data, and returns this context instance to the business logic module. Then, the business logic module invokes the method "generate_interface" in the interfaces modules passing the context instance as parameter, which returns the actual concrete interface. Finally, the business logic module returns this result to the user.

4.3 Implementation Architecture

Synth is implemented with Ruby on Rails, an MVC framework for web applications development. Within the MVC architecture, it maintains a modular organization where each module presented in the conceptual view shown in Figure 8 is implemented as a composition of one or more components of the MVC general implementation architecture.

All data in Synth is maintained as an RDF graph. It includes not only the instances of domain data, but also the metadata about the models and meta-models of SHDM, expressed as RDF(S) and OWL. This data is manipulated programmatically using the Jena framework as an API. It plays the role of the storage, inference and query layer in the persistence module in conceptual architecture. ActiveRDF [7], is a library for accessing RDF from Ruby programs, mapping the RDF data into Ruby primitives, playing the role of the RDF(S)/OWL mapping layer. ActiveRDF provides an API that facilitates CRUD operations within Ruby programs, and has an adapter for Jena.

Our previous experience (see [6]), and Oren et al [7], explain the reasons we have chosen Ruby and Rails as the implementation environment.

4.4 Authoring Environment GUI

Synth provides an authoring environment GUI using HTML forms that can be accessed from a web browser and allows the creation and editing of primitives of SHDM models. It is possible run the application while it is being built using this interface, validating it in each step of the development process.

Synth also provides the RDF Scaffold, a generic RDF browser and editor that allows the execution of CRUD operations over the local application RDF database, in the same way that it can be done in some well known RDF browsers and wikis like Tabulator, Disco[20] or OntoWiki.

4.5 DSLs within Synth

Synth provides several ways to specify the selection rules that determine the set of resources that compose a context; these rules are expressed as context query expressions. These query expressions can be specified in three alternative query languages: directly in SPARQL; in a DSL hosted in Ruby which is defined by the ActiveRDF framework; or in SynthQL, a customized simplified query language created specifically for Synth, which can represent the most common context queries in typical LDAs.

The accepted SPARQL syntax is the same accepted by the ARQ query machine, part of Jena framework, which supports the Federated SPARQL query that is important for specifying datasets in the context queries. The ActiveRDF DSL is similar to the one described in [6]

The goal of SynthQL context query language is to abstract the syntax of SPARQL for the most commons query expressions typically found in LDAs. This approach tries to minimize the need for knowledge about SPARQL, while still covering a large set of common expressions with a simple syntax. The particular DSL is similar to the one presented the previous version of Synth, HyperDE [6].

[20] http:// www4.wiwiss.fu-berlin.de/bizer/ng4j/disco/

The code below is a context query expression in SynthQL. This query means "All resources whose rdf:type is foaf:Document, dc:title contains a substring "owl", foaf:maker has the same value as the parameter 'person' passed during the user navigation, ordered by dc:title, limited to 100 results, and the query should be evaluated on the datasets identified by "local" and "iswc2010".

```
selects {
  type FOAF::Document
  dc::title like "owl"
  foaf::maker person
  order DC::title
  limit 100
  datasets :local, :iswc2010}
```

4.6 Example Application

This section shows how the example LDA described in Section 1 is implemented in Synth. For reasons of space, we will not detail the Interface Model for this application; suffices to say that the example interfaces shown there use the default interface generated by Synth for any application.

The raw RDF data from http://data.semanticweb.org/conference/iswc/2010/complete was imported to the Synth local database. This data is the core Domain Model of the application, which is enriched with the customized schedule.

After the data importing, the navigation specification was entered into Synth through its GUI (see Figure 10). The code below shows the specification of the context swc:AcademicEvent "byUserProgram".

```
:byUserProgram a shdm:Context ;
  shdm:context_name "byUserProgram ";
  shdm:context_title "Events by User Program";
  shdm:context_query "user.my::user_program.my::events";
  shdm:context_parameters [ a SHDM::ContextParameter;
                    shdm:context_parameter_name 'user'];
```

This specification uses the DSL hosted in Ruby defined by the ActiveRDF API. The Figure 10 shows the Synth interface to edit this specification. The resulting context navigation screen is shown Figure 4.

The list below is part of the specification of the "EventsByUserProgram" index. In this example some navigational attributes are omitted to save space, but they are very similar to the navigational attributes shown above.

```
:EventsByUserProgram a shdm:ContextIndex ;
  shdm:index_name "EventsByUserProgram";
  shdm:index_title "My Program";
  shdm:context_index_context :byUserProgram;
  shdm:context_anchor_attributes [
    a shdm:ContextAnchorNavigationAttribute;
    shdm:navigation_attribute_name "label";
    shdm:navigation_attribute_index_position 1;
    shdm:context_anchor_label_expression "self.rdfs::label";
    shdm:context_anchor_target_context :byUserProgram;
    shdm:context_anchor_target_node_expression "self"];
```

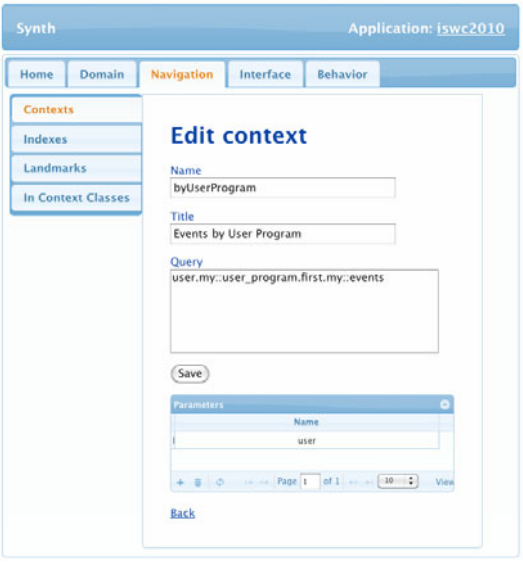

Fig. 10. Synth interface for "byUserProgram" context specification

This index is based on the context "byUserProgram" and has the navigational attribute that is a context anchor. This context anchor shows the rdfs:label of the swc:AcademicEvent and the target is the resource in which the current index entry is based on the context "byUserProgram".

5 Conclusion and Future Work

We have presented the evolution of SHDM, and Synth, a new development environment that supports the design and implementation of LDA, mixing external and locally created RDF data. The major changes in SHDM are: Domain Model defined by any RDF graph, including graphs distributed over several repositories; a Business Logic Model that unifies Hypertextual Navigation and other application functionalities; and a Hypertextual Navigation Model as an extension of any RDF graph.

Future work will add new models to SHDM, and their corresponding integration in Synth, such a authorization, transactions, and adaptation.

Acknowledgement. The authors were partially supported by grants from CNPq and Petrobras.

References

1. Bizer, C., Heath, T., Berners-Lee, T.: Linked Data – The Story so Far. International Journal on Semantic Web and Information Systems 5(3), 1–22 (2009)
2. Ceri, S., et al.: Designing Data-Intensive Web Applications. Morgan Kaufmann, San Francisco (2003)

3. Koch, N., Kraus, A.: The Expressive Power of UML-based Web Engineering. In: 2nd Int. Workshop on Web-Oriented Software Technology (IWWOST 2002), CYTED, Málaga, Spain, pp. 105–119 (2002)
4. Lima, F., Schwabe, D.: Application Modeling for the Semantic Web. In: Proceedings of LA-Web 2003, Santiago, Chile, pp. 93–102. IEEE Press, Los Alamitos (2003)
5. Luna, A., Schwabe, D.: Ontology Driven Dynamic Web Interface Generation. In: Proceedings of the 8th Int. Workshop on Web-Oriented Software Technologies (IWWOST 2009) in Conjunction with ICWE 2009, San Sebastian, Spain (June 2009)
6. Nunes, D.A., Schwabe, D.: Rapid prototyping of web applications combining domain specific languages and model driven design. In: Proc. 6th International Conference on Web Engineering (ICWE 2006), pp. pp. 153-160. ACM, New York (2006) ISBN 1-59593-352-2
7. Oren, Heitmann, B., Decker, S.: ActiveRDF: embedding Semantic Web data into object-oriented languages. Journal of Web Semantics 6(3), 191–202 (2008)
8. Rossi, G., Schwabe, D., Lyardet, F.: Web Application Models Are More than Conceptual Models. In: Procs. of the ER 1999, Paris, France, pp. 239–252. Springer, Heidelberg (1999)
9. Schwabe, D., Rossi, G.: An object-oriented approach to web-based application design. In: Theory and Practice of Object Systems (TAPOS), Special Issue on the Internet, vol. 4#4, pp. 207–225 (October 1998)
10. Silva de Moura, S., Schwabe, D.: Interface development for hypermedia applications in the semantic web. In: Proc. WebMedia and LA-Web, 2004, Ribeirão Preto, Brazil, pp. 106–113. IEEE Press, Los Alamitos (2004)
11. Thomas, D., Barry, B.M.: Model Driven Development: The Case for Domain Oriented Programming. In: Companion of the 18th OOPSLA, pp. 2–7. ACM Press, New York (2003)
12. Van Deursen, A., Klint, P., Visser, J.: Domain Specific Languages: An Annotated Bibliography, `http://homepages.cwi.nl/~arie/papers/dslbib/`
13. Vdovjak, R., Frasincar, F., Houben, G.J., Barna, P.: Engineering Semantic Web Information Systems in Hera. Journal of Web Engineering 2(1&2), 3–26 (2003)