Rapid Prototyping of Web Applications combining Domain Specific Languages and Model Driven Design

Demetrius Arraes Nunes
Departamento de Informática, PUC-Rio
Rua M. de S. Vicente, 222
Rio de Janeiro, RJ 22453-900, Brazil
+55 21 2521 2848
dema@tecgraf. puc-rio.br

Daniel Schwabe
Departamento de Informática, PUC-Rio
Rua M. de S. Vicente, 222
Rio de Janeiro, RJ 22453-900, Brazil
+55 21 3114 1500 x4356
dschwabe@inf. puc-rio.br

ABSTRACT

There have been several authoring methods proposed in the literature that are model based, essentially following the Model Driven Design philosophy. While useful, such methods need an effective way to allow the application designer to somehow synthesize the actual running application from the specification. In this paper, we describe HyperDE, an environment that combines Model Driven Design and Domain Specific Languages to enable rapid prototyping of Web applications.

Categories and Subject Descriptors

H.5.4 [**Hyperext/Hypermedia**]: Architecture; Navigation. D.2.1 [**Software Engineering**]: Requirements/Specification. D.2.2 [**Software Engineering**]: Design Tools and Techniques.

General Terms

Design, Languages.

Keywords

Model-based Designs, Hypermedia Authoring.

1. INTRODUCTION

There have been several methods for Web application design proposed in the literature, such as OOHDM [4], SHDM [3], and others. Remarkably, they all follow the principles of Model Driven Design (MDD) [5]. Simply stated, this approach uses the notion of models to help the designer perform the design activity.

Here, we show how the HyperDE environment (freely available at http://server2.tecweb.inf.puc-rio.br:8000/hyperde) supports the rapid prototyping of Web applications through a combination the Model Driven Development approach with the use of Domain Specific Languages (DSL's) [6]. This combination allows the designer/developer to write code by directly manipulating the models that specify the application. In addition, since the model is specified following the meta-model for a method, it also possible to dynamically manipulate the model itself during execution, enabling very concise and general applications. Consequently, scripts in the generated DSL work as very high level procedural specifications of the application. Our goal is to show how MDD can be combined with DSLs to provide principled, sistematic rapid prototyping of Web (and hypermedia) applications.

2. SHDM Meta Model

Figure 1 shows the SHDM meta model, with the main classes highlighted. The class NavClass models the navigation nodes, and the class Link models the links between them. Each NavClass has NavAttributes, NavOperations and Links, and can be a specialization of a BaseClass. Contexts are sets of objects of NavClass, defined through a query specified in one of its attributes; this query may have a parameter. Indexes are made out of IndexEntries, which contain either anchors to other indexes or anchors to elements within a context. Landmarks are anchors to either Indexes or to Context elements. Views allow exhibiting the contents of NavClass instances within some context, or exhibiting Indexes. Designing a Web application using SHDM corresponds to instantiating this metamodel. The HyperDE environment supports this.

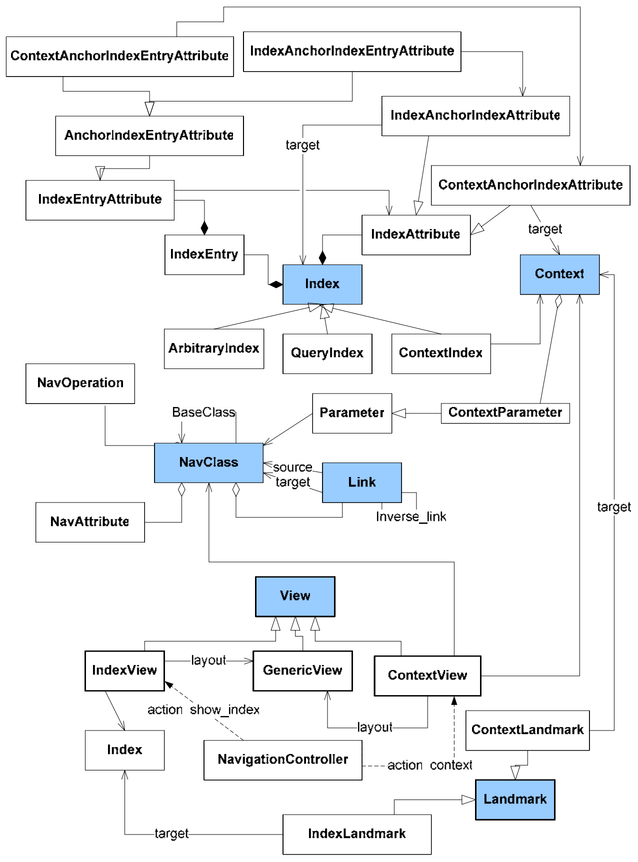

Figure 1. SHDM Meta Model

3. THE ARCHITECTURE OF HYPERDE

The HyperDE environment is based on the MNVC framework [2], which extends the MVC framework with navigation primitives. It allows the designer to input SHDM navigational models (the "model" in the MVC framework), and interface definitions (the "view" in the MVC framework), and generates complete applications adherent to the specification. It also provides an interface to create and edit instance data, although, strictly speaking, this should actually be part of the generated application. Figure 2 shows the architecture of HyperDE.

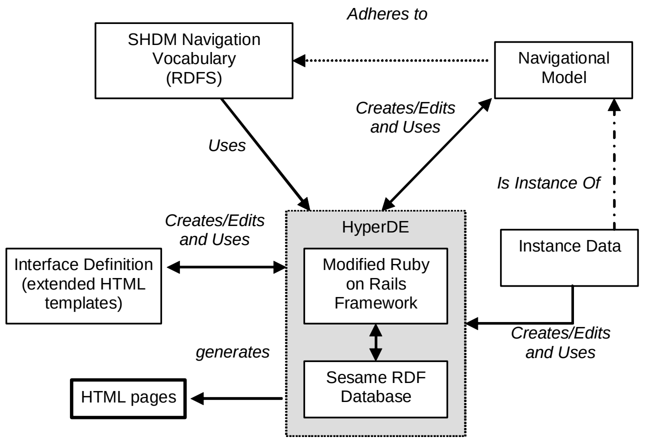

Figure 2. The architecture of the HyperDE environment.

HyperDE is implemented as a modification of the Ruby on Rails framework (http://www.rubyonrails.com), where the persistence layer (ActiveRecord) has been replaced by another one based on the Sesame RDF database. The SHDM meta-models, the user defined navigation models, as well as the application instance data, are all stored as RDF data.

All HyperDE functions can be accessed via Web interfaces. In addition, HyperDE also generates a Domain Specific Language (DSL) as an extension of Ruby, allowing direct manipulation within Ruby scripts of both the model and SHDM's meta-model. extend the DSL to other query boiler plate templates (e.g., X by Y by Z).

4. THE USE OF DSL's IN HYPERDE

We have argued that, regardless of the abstraction level of the specification language used to specify an application, there are portions which typically will require the designer/implementer to write some kind of code, such as for the business logic or for retrieving or storing values that flow in the interface.

In most environments, the language of choice for implementation is a programming language that is directly executable in the desired target environment. In such cases, these programs must manipulate the model's representation in terms of the programming language primitives, adding a layer of detail that is cumbersome at best. An alternative for this is to generate a DSL that makes the datatypes of the model also be the datatypes of some programming language. The advantages of this approach have already been argued in [1].

Following this approach, we have defined a DSL on top of Ruby, based on the SHDM model and metamodel, in the following way

1. Each instance of NavClass becomes a Ruby class;
2. Each NavAttribute of NavClass becomes an attribute of the corresponding Ruby class. In addition, a method "find_by_xxx" is also defined, allowing to search objects of this class according to values of this attribute; (Actually, a whole family of "find_by_xxx" and "find_all_by_xxx" methods are created dynamically allowing one to write such expressions as Professor.find_by_name_and_research_area or Student.find_all_by_year_and_department)
3. Each NavOperation of NavClass becomes a method of the corresponding Ruby class;
4. Each link having NavClass as the source type becomes an attribute of special type "Array", whose elements are objects of the target type of the link.

The built-in Ruby operators are redefined to handle the expected semantics. For instance, if a new element is inserted in the array that corresponds to a link, this is interpreted as creating a new link instance. Consider for instance the following code

```
schwabe = Professor.find_by_name "Daniel Schwabe"
area = ResearchArea.find_by_name "Hypermedia"
for student in schwabe.advises
  unless student.works_in.include?(area)
    student.works_in << area
  end
end
```

This code ensures that for all students advised by *professor* "Daniel Schwabe" have a *student.works_in* relation to the "Hypermedia" *researchArea*. This illustrates the conciseness of the code using the generated DSL.

Acknowledgement. Daniel Schwabe was partially supported by a grant from CNPq, Brazil.

5. REFERENCES

[1] Goldman, N. M. Ontology-Oriented Programming: Static Typing for the Inconsistent Programmer, Lecture Notes on Computer Science - The Semantic Web - ISWC 2003, Springer-Verlag Heidelberg, Volume 2870 / 2003 - Outubro, 2003, pp. 850-865

[2] Jacyntho, M. D., Schwabe, D. , Rossi, G. A software architecture for structuring complex web applications. Journal of Web Engineering, Vol 1, No 1, (2002).

[3] 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).

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

[5] Thomas, D., Barry, B.M.; "Model Driven Development: The Case for Domain Oriented Programming", Companion of the 18th OOPSLA, ACM Press, 2003, pp. 2-7.

[6] Van Deursen, A.; Klint, P.; Visser, J.; "Domain Specific Languages: An Annotated Bibliography", http://homepages.cwi.nl/~arie/papers/dslbib