Model Driven Prototyping with Modelibra

Dzenan Ridjanovic

Université Laval, 2325 rue de la terrasse,
Québec (Québec) G1V 0A6, Canada
dzenanr@gmail.com

Abstract. A domain model may be designed in iterations, where each iteration is validated through a default application generated from the model. By adding, editing, removing, retrieving and navigating real data it becomes obvious what is wrong with the model. Once the model is validated, user views of the model may be developed by reusing generic components or by developing specific components. Modelibra is a software family of tools and frameworks that supports this prototyping process of designing domain models and their views.

Keywords: domain models, model frameworks, model validation, model prototyping, model views.

1 Introduction

A domain model is a model of specific domain classes that describe the core data and their behavior [1]. In business terms, a domain model is a model of the domain. Within the domain, an organization conducts its business [2]. The memory of any organization may be abstracted as a domain model. The backbone of data intensive software is a domain model. The core of an organizational information system is a domain model.

It is hard to design a model that reflects well its domain, as it is not easy to develop software based on a domain model. A way to validate and improve a domain model is to use an application developed on the model. It is much easier for a domain expert to use the application than to read the model. By adding, editing, removing, retrieving and navigating real data it becomes obvious to a domain expert what is wrong with the model. However, it takes resources to develop an application based on a domain model. A research goal is to reduce the time to develop an application based on a domain model.

There are two categories of prototypes developed on a domain model. The first category represents applications that are very close to their domain models. Its objective is to make a domain model alive to allow users to validate the model. After the model is validated, the application is pushed to the background to allow an administrator to quickly make some changes in data. This category represents model prototypes. A prototype of a domain model may be considered as a default application of the model. After some improvements in the model, a new prototype is constructed to continue with model validations. This is repeated until the model becomes stable.

There are some simple rules that determine how a domain model may behave by default as an application. Some of those rules may be reconfigured to make a default application a bit easier to use. The rules may be used to even generate a default application based on a domain model and its configuration.

The second category of prototypes represents applications that are close to user views of the domain model [3]. Its objective is to validate user views of the domain model. After views are validated, the application may be used to satisfy some basic needs of users. This category represents functional prototypes. A functional prototype may be considered as live design of user interfaces. In order to quickly respond to user demands, a catalogue of reusable generic components must be available. A generic component uses the domain model, but is not dependent on specific details. A decent size catalogue of generic components increases development productivity.

One of bottlenecks of model driven development is a database that represents a domain model. It is time consuming to propagate changes of a domain model to a database schema. If our database is relational, it is not trivial to develop a code that would map objects in main memory to table rows in a database.

2 Modelibra Software Family

Modelibra [4] is open source software (OSS) family that is designed to enable model driven prototyping. It consists of tools and frameworks, developed in Java, which support the design of domain models, the code generation of model configurations, the code generation of default applications, and the component based development of web pages. By default, Modelibra uses XML data files to persist user data and not relational databases. In this way, there are no complicated installations and difficult mappings. It is easy to design graphically a domain model, to generate its XML configuration, to generate a default application, to validate the model by entering some real data, to improve the domain model and regenerate the code. A developer of Modelibra is able to show you in an hour how to go through this process for a small domain model with a few concepts and relationships.

In addition to prototypes, Modelibra is well suited for small applications that may have a relatively complex domain model but not that many data. The main restriction of Modelibra is the requirement that all data must be present in main memory. This allows software developers to focus on objects in a pure object oriented way and completely ignore specifics of relational databases. In future, Terracotta [5] will be integrated into Modelibra to expand the use of Java virtual machine from a single computer to multiple computers connected via Internet or Intranet. A software developer would use Modelibra as if there was only one computer. It would be a task for Terracotta to reach objects in different computers, extending the main memory of a single Java virtual machine to the Web memory. Of course, everything should be done to prevent a loss of data in main memory. Modelibra uses XML representation to save data from the main memory to files.

In prototype applications or small OSS projects there is often only one developer. Similarly, in educational context, students work on individual assignments. Their projects may use some complex domain models but usually do not require large

quantities of data. Modelibra has been used in university education for several years [6].

The Eclipse Modeling Framework (EMF) [7] belongs to the same software category as Modelibra. EMF is a Java framework and code generation facility for building tools and other applications based on a structured model. It is a professional framework that may not be easy to learn for developers of prototypes or small applications and students of software engineering or information systems.

3 Domain Model

A domain model consists of user concepts, concept properties and relationships between concepts, which are represented graphically. Figure 1 represents a domain model of web links that are of interest to certain members. The domain model has been designed with the help of the ModelibraModeler tool. The domain is called DmEduc and the model is named WebLinks.

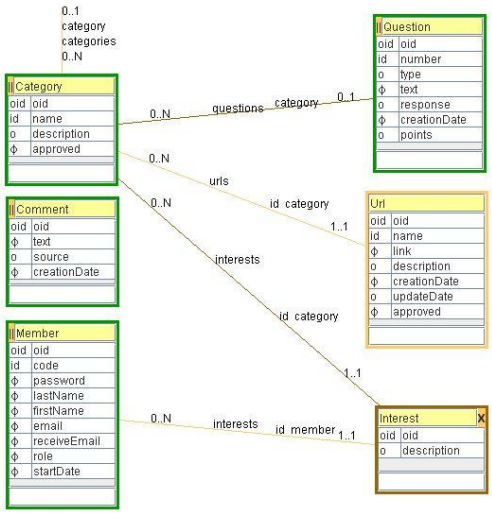

Figure 1. WebLinks model in DmEduc domain

The domain model's concepts are: Url (Uniform Resource Locator), Question, Category, Member, Interest and Comment. The Url concept describes a web link. Urls are divided into categories that may have subcategories. The Question concept represents a frequently asked question about the use of the web application. Questions are optionally categorized. Members show their interests in categories of web links. Comments can be made about anything related to the web application.

A concept is defined by its attributes: properties for data and neighbors for relationships. For example, the Url concept has the description property, which may be null (the o symbol in front of the property name). A relationship between two concepts is represented by two neighbor directions displayed together as a line. A neighbor direction is a concept's special property, with a name and a range of

cardinalities. Note that both neighbor name and cardinalities are displayed close to the source concept, which is different from the notation used in UML [8], where cardinalities are moved close to the destination concept. Since the neighbor name and cardinalities are neighbor properties of the source concept, it is more natural to use them in a similar manner as properties. A concept's neighbor is either a child or a parent. A child neighbor has the maximum cardinality of N (or a number greater than 1). A parent neighbor has the maximum cardinality of 1.

At the level of data, a concept is represented as a collection of entities. The retrieval of entities starts with the entry concepts of the domain model. In the WebLinks domain model, the entry concepts are Category, Question, Member and Comment. They all have a green border and the || symbol in the upper left corner of the concept. Once an entity of the entry concept is retrieved in the collection of entities, the retrieval of neighbor entities may start. A child neighbor is represented as a collection of entities. A parent neighbor is represented as a single entity.

The Url concept is not an entry concept. Hence, entities of the Url concept may be reached only through its parent Category concept. As a non-entry, the Url concept has a light orange border. The Interest concept has two parents. Thus, interests may be retrieved either from the Member concept or the Category concept. A concept that has more than one parent is called an intersection concept. An intersection concept that represents a many-to-many relationship has a brown border and the X sign in the upper right corner of the concept.

Every concept has a predefined property called oid. The oid property is mandatory. It is used as an artificial concept identifier and is completely managed by Modelibra. Its value is unique universally. In addition, a concept may have at most one user oriented identifier (id), which consists of properties and/or neighbors. In an entry concept, all entities must have a unique value for the concept id. However, in a non-entry child concept, the id is often unique only within the child parent. For example, the id of the Url concept is defined by the name property and the category neighbor. Hence, a name must be unique only within its category.

By default, a neighbor is internal. In that case, a relationship has a light orange color. Starting with an entry concept and following internal neighbors, produces a hierarchical submodel of concepts. By default, this submodel of entities is saved in a single XML data file. When a neighbor is external, the relationship has a brown color and it connects two concepts, each in a different hierarchical submodel. The child entity has a reference oid that points to the parent entity. The reference oid is used by Modelibra to load data from XML data and to reconstruct a pure object model of entities out of submodels of entities.

4 Modelibra Default Application

A domain with its models must be configured in an XML configuration file. Fortunately, this configuration may be generated from a graphical domain model. The XML configuration is used to generate the model's POJO [9] classes in Java. However, the XML configuration provides more information about the model's

default behavior used heavily in generic components. The XML configuration is loaded up-front by Modelibra and converted into meta entities in the main memory.

Modelibra has a Java framework for default applications called ModelibraWicket. ModelibraWicket uses Wicket [10] web framework for the construction of web components. ModelibraWicket interprets a domain model as a web application. We will show here only a few web pages of a default web application of the DmEduc domain.

The home page (Figure 2) of the application provides a button-like link to enter Modelibra. There are two other links that are not active yet. The DmEduc domain name is displayed in the the title section.

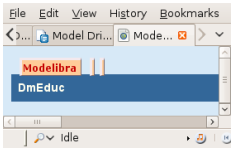

Figure 2. Home page

The Modelibra page (Figure 3) is the application's domain page with a table of domain models. The domain title is DmEduc Domain and the only model in the domain is WebLink. The model has two button-like data links, one for displaying model entries and another for updating model entries. The Home link displays the application home page.

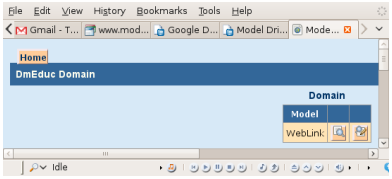

Figure 3. Modelibra page

The display model page (Figure 4) presents a table of model entry concepts. Every entry concept has Display and Select links. The Display link shows the concept as a table of entities. Only required properties are presented.

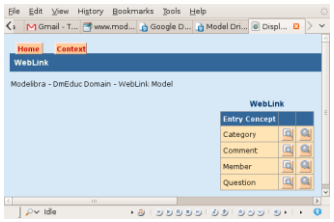

Figure 4. Display model page

The update model page presents a table of model entry concepts with Update links. Each entry concept has its own display and update page with a table of entities. The Context link in the upper left corner of a page plays a role of the back button. An entity may be displayed, selected or updated. A new comment can be added and an

existing comment can be displayed (with all details), edited or removed. In Figure 5, only one comment is displayed.

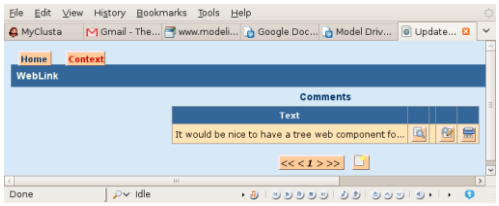

Figure 5. Update table of comments

When an entity is added or edited using a form, its properties are validated with respect to the XML configuration of the concept. If there are errors, they are displayed in front of the form. For example, in Figure 6, an empty comment is added. When the add form is displayed, the Save button is clicked to add an empty comment. Since the text property is required, the corresponding error is displayed in front of the form, and a small star after the field in question shows that there is a problem with the value of that field.

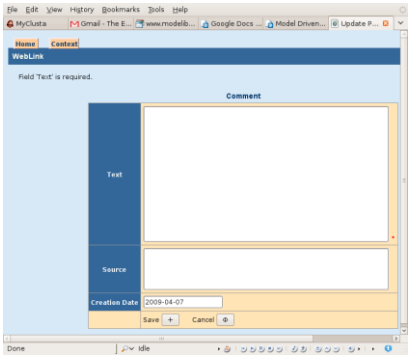

Figure 6. Add comment form

In Figure 7 only Software categories are shown. Note that both parent and child categories are displayed. For the Software parent category there are three subcategories: Framework, Tools, Web Meeting. Each displayed category has its corresponding Urls (web links) and Categories (subcategories).

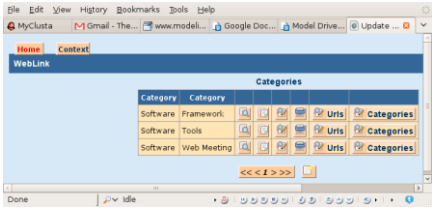

Figure 7. Category subcategories

Web links of the Framework category are shown in Figure 8.

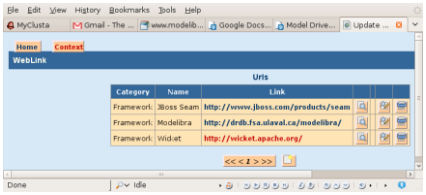

Figure 8. Category web links

There is also a generic mechanism for making selections of entities. In Figure 9, only questions that contain the word Modelibra in the text property will be selected.

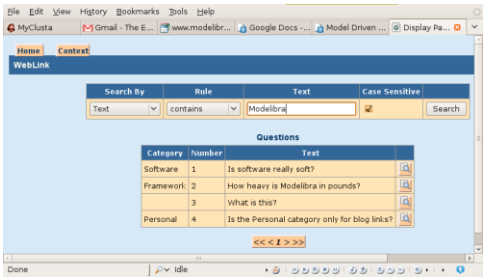

Figure 9. Selection of entities

5 Modelibra Views

Once a domain model is validated through the model's default application, model views may be used to start developing web components that would be building blocks of web pages. The new home page will be presented here, where each section of the page is a placeholder of a web component. Some of those web components will be specific and some generic. A specific component is developed by a web application programmer with the objective to focus on a specific page section. A generic component is developed by an advanced programmer to support the needs of different web applications through its parameters.

A web component, specific or generic, uses a part of a domain model, called a component view model. A web component has also its view presentation parameters. A framework of the Modelibra software family that focuses on web components is ModelibraWicket. ModelibraWicket uses the Modelibra domain model framework for its web view models, and the Wicket web framework for its web views. In the opposite direction, Modelibra does not have any knowledge of ModelibraWicket, and Wicket does not know anything about ModelibraWicket and Modelibra.

Why is Wicket used for web components? In Wicket, the development of web applications is concentrated in Java classes. For each concept of a web application there is a corresponding Java class in Wicket. For example, for a web section there is the Panel class in Wicket. XHTML and CSS are used to display a web component view in a usual way. In Figure 10, the Home page is composed of several web components.

Figure 10. Home page with web components

The HomeMenu Java class is a specific web component that is used only in this web application. The SigninPanel Java class is a generic web component that may be reused in different web applications. The SigninPanel component may be found in a catalogue of generic web components of ModelibraWicket. Both specific and generic web components are used in similar way.

The HomeMenu component is added in the HomePage specific class in the following way.

```
DmEducApp dmEducApp = (DmEducApp) getApplication();
DmEduc dmEduc = dmEducApp.getDmEduc();
WebLink webLink = dmEduc.getWebLink();
ViewModel homePageModel = new ViewModel(webLink);
View homePageView = new View();
homePageView.setWicketId("homeMenu");
add(new HomeMenu(homePageModel, homePageView));
```

The first step is to get the WebLink model. The model is obtained from the DmEduc domain that comes from the DmEducApp web application. The getApplication method is inherited from Wicket. The menu component has two composite arguments. The first argument is an object of the ViewModel class. The second argument is an object of the View class. Both classes come from ModelibraWicket. The two composite arguments are constructed and their specific parameters are set. Then, the menu component with those two filled arguments is constructed and added to the home page.

The SigninPanel generic component is added to the home page in a similar way.

```
ViewModel signInViewModel = new ViewModel(webLink);
signInViewModel.setEntities(members);
signInViewModel.setEntity(new Member(members.getModel()));
View signInView = new View();
signInView.setWicketId("signIn");
Panel signIn = new SigninPanel(signInViewModel, signInView);
add(signIn);
add(new FeedbackPanel("signInFeedback"));
```

```
if (getAppSession().isUserSignedIn()) {
  signIn.setVisible(false);
}
```

The view model consists of a collection of existing members and a member object that will accept two values entered by a user. If a member with the given code and password values exists, the sign in will be successful and the sign in panel will become invisible. If there is an error, the error message will be displayed in a feedback panel and the sign in panel will stay visible to allow a user to enter correct values.

The Comments component is defined on the single Comment concept based on the current entities of that concept. The display is of the table type and the concept's essential properties are displayed, here only the text property. The class of the generic web component is EntityDisplayTablePanel.

```
ViewModel commentsModel = new ViewModel(webLink);
Comments comments = webLink.getComments();
commentsModel.setEntities(comments);
View commentsView = new View();
commentsView.setWicketId("commentTable");
EntityDisplayTablePanel commentTable = new
  EntityDisplayTablePanel(commentsModel, commentsView);
add(commentTable);
```

The Comments web component has a simple view model based on one concept. Often there is a need to display data based on the one-to-many relationship between two concepts. The ParentChildPropertyDisplayListPanel generic component displays one property from a parent entity and one property from a child entity, in a list of parents with the corresponding child list for each parent.

```
ViewModel categoryUrlsModel = new ViewModel(webLink);
Categories categories = webLink.getCategories();
Categories orderedApprovedCategories = categories
  .getApprovedCategories().getCategoriesOrderedByName();
categoryUrlsModel.setEntities(orderedApprovedCategories);
categoryUrlsModel.setPropertyCode("name");
categoryUrlsModel.getUserProperties().addUserProperty(
  "childNeighbor", "urls");
categoryUrlsModel.getUserProperties().addUserProperty(
  "childProperty", "link");
View categoryUrlsView = new View();
categoryUrlsView.setWicketId("categoryNameUrlLinkList");
categoryUrlsView.setTitle("Category.WebLinks");
ParentChildPropertyDisplayListPanel categoryNameUrlLinkList =
  new ParentChildPropertyDisplayListPanel(categoryUrlsModel,
    categoryUrlsView);
add(categoryNameUrlLinkList);
```

In the view model, the categories entry point is obtained from the domain model and its subset of only approved categories, ordered by name, is set as entities. The

property code is set to name. The ViewModel class has a generic way of adding additional, user oriented properties. Two user properties are defined here and they are used to find children of a parent and a child property that will be displayed. The childNeighbor user property is the urls neighbor of the Category concept. The childProperty user property is the link property of the Url concept.

6 Conclusion

Modelibra is a software family that provides tools and frameworks to design a domain model, to generate Java code for the model, to validate the model through a default application, and to use generic components, based on the model, for user views. It is hard to design well a domain model. A way to help designers is to generate a default application based on the model and to use the application in order to realize what needs to be done to improve the model.

Once a domain model is validated, it is also important to quickly create user views of the model. This is only possible if user views are not created from scratch, but from a well designed catalogue of generic components.

References

1. DSM Forum
http://www.dsmforum.org/
2. Domain Modeling
http://www.aptprocess.com/whitepapers/DomainModelling.pdf
3. Model-driven prototyping
http://hci.uni-konstanz.de/downloads/CorporateUISpec_IFIP_TM_CB_HR.pdf
4. Modelibra
http://www.modelibra.org/
5. Terracotta
http://www.terracotta.org/
6. Modelibra Education
https://www.assembla.com/wiki/show/modelibra
7. Eclipse Modeling Framework
http://www.eclipse.org/emf/
8. Unified Modeling Language
http://www.uml.org/
9. Plain Old Java Objects
http://en.wikipedia.org/wiki/POJO
10. Wicket
http://wicket.apache.org/

Ridjanovic, D., "Model Driven Prototyping with Modelibra", *The Second International Workshop on Internet Engineering & Web Services*, InWeS-2011, Ankara, Turkey, http://aircccse.org/inwes2011/inwes2011.html, June 26 - 28, 2011.