

Modelibra Software Family

Dzenan Ridjanovic
Université Laval
Québec G1K 7P4, Canada

Abstract -This paper provides a brief overview of Modelibra, the open source software family that is used to develop dynamic web applications based on domain models. The software family consists of a graphical design tool, a domain model framework, a web component framework, a collection of CSS declarations, and XML, database and Java code generators. Modelibra facilitates the definition and the use of domain models in Java. It uses Wicket for application views of a domain model. Wicket is a web framework that provides web components to construct, in an object oriented way, web concepts, such as web pages and page sections. Modelibra interprets the application model and creates default web pages based on the model. A default application may help developers validate and consequently refine the domain model. In addition, Modelibra has a collection of generic web components that may be easily reused in professional web applications to display or update entities.

I. MODELIBRA

In computer terms, a domain model is a model of specific domain classes that describe the core data and their behaviour [1]. The heart of any software is a domain model. When a model is well designed and when it can be easily represented and managed in an object oriented language, a developer may focus on views of the software and they are what users care about the most.

Modelibra has been designed to help developers in representing and using application domain models in a restricted way. The main restriction of Modelibra, and at the same time its main feature, is that all data must be present in main memory. This and other restrictions of Modelibra minimize the number of decisions that a domain designer must make. This makes Modelibra easy to learn. Modelibra is an Open Source Software (OSS) [2]. It is hosted at JavaForge [3]. Developers of OSS may find Modelibra useful for developing their software around a domain model, for providing an easy installation for their users and for developing a web application to introduce their software to the public audience.

By default, Modelibra uses XML files to save model objects. However, Modelibra allows the use of both relational and object databases. The upgrade of an application from XML data files to a database does not require a single line of programming code to be changed. Modelibra also provides the data migration from XML files to a database. Although the focus of Modelibra is a software application with relatively small amount of data, it provides some advanced features such as transactions and undo. A domain may have several models. One of them may be a reference model where common data, common to all models within a domain, are kept. A domain model may also inherit some of its definitions from another model in the same domain. In Modelibra, a part of the base

model may be exported to another model, which can be taken for an off-line work, then returned back to synchronize changes with the base model.

The Modelibra software family consists of a graphical design tool, a domain model framework, a web component framework, a collection of CSS declarations, and code generators for XML configurations, database schemas and Java classes. Modelibra facilitates the definition and the use of domain models in Java. Modelibra interprets the application model and makes it alive as a default web application, which may help developers validate and consequently refine the domain model.

The software closest to Modelibra is Eclipse Modeling Framework (EMF) [4]. The objective of EMF is to provide code generation facility for building tools and other applications based on a structured data model. It has additional components for queries, transactions, model integrity validation and service data objects. It is quite a complex software that is not that easy to learn.

The next section will start with a simple domain model, which will be used to show how a web component is built from the model.

II. DOMAIN MODEL

A domain model is a representation of user concepts, concept properties and relationships between concepts. The easiest way to present a domain model is through a graphical representation. Fig. 1 represents a domain model of a simple web application called Web Links. It features web links that are of interest to certain members.

In our case, the domain model's concepts are: Url, Question, Category, Member, Interest and Comment. Url describes a web link. Urls are categorized. Categories are organized in a tree of subcategories. Question is a frequently asked question about the use of the web application. Questions are optionally categorized. Members express their interests in categories of web links. Comments can be made about anything related to the web application.

A concept is described by its properties and neighbors, called together the concept's attributes. The Url concept has only one neighbor, the Category concept. However, the Category concept has four neighbors: Url, Question, Interest and Category concepts. A relationship between two concepts is represented by two neighbor directions, displayed together as a line. A neighbor direction is a concept special (neighbor) property, with a name and a range of cardinalities. A neighbor

is either a child or a parent. A child neighbor has the max cardinality of N (or a number greater than 1). A parent neighbor has the max cardinality of 1. If a parent neighbor has the min cardinality of 0, the parent is optional.

A concept is represented as a list of entities. The retrieval of entities starts with the entry concepts of the domain model

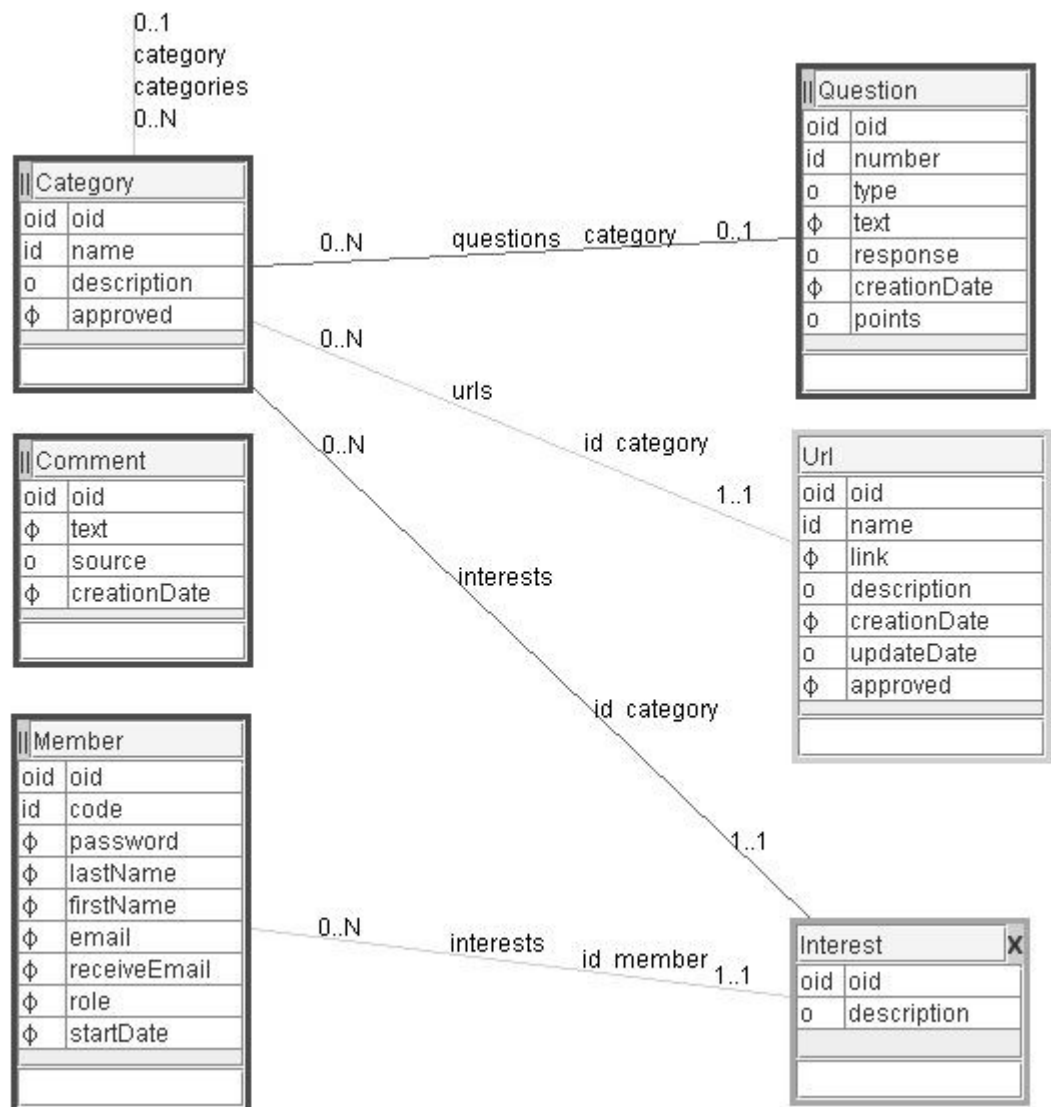


Fig. 1. Web Link domain model

In the Web Link domain model, the entry concepts are Category, Question, Member and Comment. They all have a darker border and the || symbol in the upper left corner of the concept. Once an entity of the entry concept is retrieved in the list of entities, the retrieval of neighbor entities may start. A child neighbor is represented as a list 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. 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 and has 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 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) that consists of the concept's properties and/or neighbors. A simple id has only one property. 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.

A graphical design tool called ModelibraModeler was used to create the Web Link domain model. The tool generates a relational database schema and an XML configuration of the domain model. Modelibra reads the XML configuration and uses it quite often in interpreting the model to support different actions on the model. The following is a short excerpt of the domain configuration for the Web Link model.

```
<domain oid="1101453345237">
  <code>DmEduc</code>
  <type>Specific</type>
  <models>
    <model oid="1101453347786">
      <code>WebLink</code>
      <author>Dzenan Ridjanovic</author>
      <concepts>
        <concept oid="1101453347834">
          <code>Category</code>
          <entitiesCode>
            Categories
          </entitiesCode>
          <entry>true</entry>
          <properties>
            <property oid="1101453347889">
              <code>name</code>
              <propertyClass>
                java.lang.String
              </propertyClass>
              <required>true</required>
              <unique>true</unique>
            </property>
            ...
          </properties>
          <neighbors>
            <neighbor oid="1101453348237">
              <code>urls</code>
              <destinationConcept>
                Url
              </destinationConcept>
              <type>child</type>
              <min>0</min>
              <max>N</max>
            </neighbor>
          </neighbors>
        </concept>
        ...
      </concepts>
    </model>
  </models>
</domain>
```

By default, each entry point into the model is saved in its XML data file. The following is a part of categories data.

```
<categories>
  <category oid="1193171129084">
    <name>Software</name>
    <approved>true</approved>
    <urls>
      <url oid="1194575051451">
        <name>
          Open Source Software in Java(tm)
        </name>
        <link>http://java-source.net/</link>
        <creationDate>2007-11-08</creationDate>
        <approved>true</approved>
      </url>
    </urls>
  </category>
  <category oid="1193171173850">
```

```
<name>Framework</name>
<approved>true</approved>
<urls>
  <url oid="1193171279804">
    <name>Wicket</name>
  ...
```

III. ENTITIES

A concept is represented in Modelibra as two Java classes, one to represent an entity and the other to represent a list of entities, both Plain Old Java Objects (POJOs) [5]. For example, the Category concept has two classes: Category and Categories. The Category class extends the GenCategory class. The Categories class extends the GenCategories class. The generic classes are generated by Modelibra from the XML configuration. The specific classes, also generated the first time by Modelibra, are almost empty. A developer may add a specific code to the specific classes, which will not be lost when the model changes and when the generic classes are regenerated.

```
public class Category extends GenCategory
public class Categories extends GenCategories
```

The abstract GenCategory class extends the Entity class passing the Category class as a generic type parameter. Similarly, the abstract GenCategories class extends the Entities class.

```
public abstract class GenCategory extends
  Entity<Category>
public abstract class GenCategories extends
  Entities<Category>
```

The abstract Entity class implements the IEntity interface. The abstract Entities class implements the IEntities interface.

```
public abstract class Entity<T extends IEntity>
  extends Observable implements IEntity<T>
public abstract class Entities<T extends IEntity>
  extends Observable implements IEntities<T>
```

Modelibra has an Eclipse project skeleton, called ModelibraWicketSkeleton, where the XML configuration is generated from ModelibraModeler. The skeleton has predefined directories for a Wicket web application with a collection of CSS declarations for a web page layout. In addition to Java classes, different files, such as XML data files and property files with display text, are also generated to make a complete web application.

IV. WEB COMPONENTS

A web application is a collection of web pages. A dynamic web application has some pages that are generated dynamically based on some data. When those data change, the content of the corresponding web pages changes as well.

A web framework, called Wicket [6], is used to compose a default web application from the domain model. Wicket is a

web application framework for creating dynamic web pages by using web components for web application concepts such as web pages and page sections. It uses only two technologies: Java and HTML. Wicket pages can be designed by a visual HTML editor. Dynamic content processing and form handling is all handled in Java code.

Modelibra makes a domain model alive as a web application, so that model data may be displayed as web pages and updated through forms. The model is metamorphosed into a web application with the help of the XML configuration by a web framework of Modelibra that is called ModelibraWicket. With some small changes in the XML configuration, the web application may be somewhat customized. It is important to realize that this web application is not a version that one would like to install as a web site. Its main purpose is to validate a domain model by designers and future users of the web application and consequently refine the domain model. In addition, ModelibraWicket has a collection of generic web components that may be easily reused to customize the web application.

In order to construct a web component, which represents a section of a web page, two component arguments must be prepared. In order to prepare those two arguments the domain model must be reached from the web application.

```
DmEducApp dmEducApp = (DmEducApp) getApplication();
DmEduc dmEduc = dmEducApp.getDmEduc();
WebLink webLink = dmEduc.getWebLink();
```

The WebLink model is obtained from the DmEduc domain, which is found in the DmEducApp that extends the DomainApp class from ModelibraWicket. The DomainApp class extends the WebApplication class from Wicket. Once the domain model is reached, in three lines of code, the model of the web component is determined by the ViewModel class of ModelibraWicket.

```
ViewModel commentsModel = new ViewModel(webLink);
Comments comments = webLink.getComments();
commentsModel.setEntities(comments);
```

The view model of the web component is based on the WebLink model. Its entities are Comments that are entry point into the model.

The view of the web component carries a Wicket id that connects the Java web component with its HTML code.

```
View commentsView = new View();
commentsView.setWicketId("commentTable");
```

Finally, the web component is constructed and added to its page container.

```
add(new EntityDisplayTablePanel(commentsModel,
    commentsView));
```

The web component comes from ModelibraWicket. Its name reflects its purpose. Starting with the end of the name, the

component is a **panel** that will be placed in a section of a web page. The dynamic data will be presented as a **table** of entities. The entities will only be **displayed** without possibility to update them with this web component. The web component scope is only one **entity** concept. In short, the web component will display a table of comments, originally in color but transformed in gray in this paper. Only the **text** property is shown, because it is the only essential property of the Comment concept, defined as such in the XML configuration.





Comments	
Text	
It would be nice to have a tree web component fo...	
This home page could have been generated by Mode...	
There is no much code behind web components on t...	
How do I become a member?	

Fig. 2. Comments Web Component

The corresponding HTML code for the web component is placed in an HTML page that contains the component's dynamic data. The HTML element where the data will be displayed is determined by the opening and closing div tags. The opening tag has the wicket:id attribute whose value must be identical to the id value of the web component.

```
<div wicket:id="commentTable">
To be replaced dynamically by the table of comments.
</div>
```

Another generic web component from ModelibraWicket whose scope is only one concept is called EntityPropertyDisplayListPanel. The component is a **panel** that will be placed in a section of a web page. The dynamic data will be presented as a **list** of entities. The entities will be **displayed** without possibility to update them with this web component. Only one **property** of the concept will be shown. The component's scope is one **entity** concept.

```
ViewModel questionsModel = new ViewModel(webLink);
Questions questions = webLink.getQuestions();
questionsModel.setEntities(questions);
questionsModel.setPropertyCode("text");
```

```
View questionsView = new View();
questionsView.setWicketId("questionTextList");
questionsView.setTitle("Questions");
```

The view model consists of questions and the view title is determined by the Questions key in a property file of the web application.

```
Questions=Questions
```

The web component is constructed from the prepared

arguments and added to the same web page but to a different section of the page.

```
add(new EntityPropertyDisplayListPanel(
    questionsModel, questionsView));
```

The given Wicket id is found in the corresponding HTML element.

```
<div wicket:id="questionTextList">
To be replaced dynamically by the list of questions.
</div>
```

The Java and HTML code of the web component, together with CSS declarations of the web application, gives the following content and look.

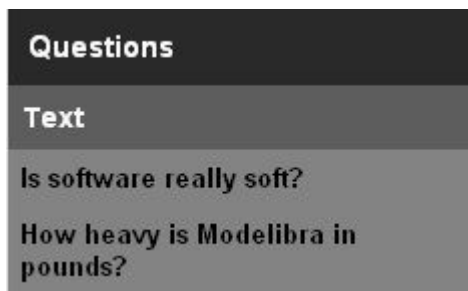


Fig. 3. Questions Web Component

The last web component shown in this paper covers two concepts and one relationship. The concepts are *Category* and *Url*. The first concept is an entry into the model, while the second concept is not. As a consequence, urls can be retrieved only through their parent category. Only approved categories, of the root level of the hierarchy of categories, will be ordered by name, which will be the only property shown. For each approved category the urls neighbor will be reached and only the link property of the neighbor concept will be displayed. In order to indicate the child neighbor and the child property two user properties are introduced.

```
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");
```

The *Category.WebLinks* key is used to determine a title of the web component.

```
Category.WebLinks=Category Web Links
```

The key with its value is located in a property file of the web application. In an international version of the web application there are different files with the same keys but different values. The following is the title of the web component in French.

```
Category.WebLinks=Liens Web des catégories
```

The web component has a rather long name, but the name indicates that component is a **panel** in which a **list** will be **displayed**. Only one **property** from both the **child** and **parent** concepts will be shown.

```
add(new ParentChildPropertyDisplayListPanel(
    categoryUrlsModel, categoryUrlsView));
```

The HTML code, with the help of Wicket id, determines a place within a web page where the component will be displayed.

```
<div wicket:id="categoryNameUrlLinkList">
To be replaced dynamically by the list of categories
each with its web links.
</div>
```

The web component displays a list of approved root categories, each with its web links. Even this web component, whose scope is larger than a single concept, is easy to prepare. Both Java and HTML contain only a few lines of code.



Fig. 4. Category Web Links Web Component

V. CONCLUSION

Modelibra is the OSS family that is used to develop dynamic web applications based on domain models. The software family consists of the ModelibraModeler graphical design tool, the Modelibra domain model framework which carries the same name as the software family to indicate its core status, the ModelibraWicket web component framework, CSS declarations in the ModelibraWicketSkeleton project, and various code generators in ModelibraModeler, Modelibra and ModelibraWicket. Modelibra facilitates the definition and the use of domain models in Java. With Modelibra, a programmer does not need to learn a complex data framework in order to create, maintain and save domain models. ModelibraWicket uses Wicket for application views of domain models.

Modelibra interprets the domain model and ModelibraWicket creates the default web application, which can be customized by the XML configuration and by adding a specific code. The web application helps developers validate and consequently refine the domain model. The generic code can be regenerated without losing the added specific code.

ModelibraWicket has a collection of generic web components that may be easily reused, within a few lines of code, in specific web pages in order to convert the default web application into a web application that responds well to user needs. If there is no generic web component in ModelibraWicket for a particular need, a specific web component may be developed by reusing Wicket components.

Publication

Ridjanovic, D., "Modelibra Software Family", *International Conference on Systems, Computing Sciences and Software Engineering*, CISSE/SCSS 2007, conducted through the Internet using web-conferencing tools, <http://cisse2007.org/>, December 3 - 12, 2007.

REFERENCES

- [1] DSM Forum
<http://www.dsmforum.org/>
- [2] Open Source Web Frameworks in Java
<http://java-source.net/open-source/web-frameworks>
- [3] JavaForge
<http://www.javaforge.com/>
- [4] Eclipse EMF
<http://www.eclipse.org/modeling/emf/>
- [5] Plain Old Java Object
<http://en.wikipedia.org/wiki/POJO>
- [6] Wicket Web Framework
<http://wicket.apache.org/>