Domain Model Driven Development of Web Applications

Dzenan Ridjanovic
Université Laval
Québec, Canada

Abstract

This paper provides a brief overview of two frameworks, Domain Model Lite and Domain Model RAD, which are used to develop dynamic web applications in a relatively short amount of time. Domain Model Lite is a framework that facilitates the definition and the use of domain models in Java. Domain Model RAD is a rapid application development framework that uses Domain Model Lite for domain models and Wicket for application views. 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. Domain Model RAD interprets the application model and creates default web pages based on the model.

Introduction

There are many Open Source Java web frameworks [1]. The most popular is Struts [2] from the Apache Jakarta Project. Struts relies more on external configuration files and less on Java code to speed up Web application development. It is an action based framework. As a consequence, the control part of Struts is rather elaborate for developers and is not suitable for rapid development of web applications.

There is a new web component based framework called Wicket [3]. A web component, such as a web page or a page section, is in the center of preoccupation in Wicket. The control part of Wicket is largely hidden from developers. Thus, Wicket is appropriate for rapid development of web application views.

A web application, as any other software has two major parts: a domain model and views. Although, a Wicket component requires a model for the component data, the actual model is outside of the Wicket realm. Wicket developers usually use Hibernate [4] to represent domain models and to persist them to relational databases. Hibernate is a complex object-relational mapping framework that uses several XML configuration files to support the mapping. However, the interaction of Wicket with Hibernate is not obvious. In addition, a Wicket developer must learn a complex framework before providing even some simple data to web components. Hence, Hibernate is not an appropriate choice for rapid application development.

I have developed a domain model framework, called Domain Model Lite or dmLite [5], to provide an easy support for small domain models, which are usually used in rapid web development. Its name reflects the framework objective to provide an easy to learn and easy to use framework. I have developed, in nine spirals, a simple web application with Domain Model Lite and Wicket [6], to allow less experienced developers to learn the basics of Domain Model Lite and Wicket quickly. In addition, I have developed a web component framework, called Domain Model RAD or dmRad [5], to produce rapidly a web application based on the given domain model.

Domain Model Lite

A domain model is a model of specific domain classes that describe the core data and their behavior [7]. 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 can then focus more rapidly on views of the software, since they are what users care about the most.

There is a class of small projects where there is no need to elaborate on different design representations, such as sequence and collaboration diagrams in UML. In a small application, a domain model is the core part of the application software. Domain Model Lite has been designed to help developers of small projects in representing and using application domain models in a restricted way. The restrictions minimize the number of decisions that a domain modeler must make. This makes Domain Model Lite easy to learn.

In most cases, domain model data must be saved in an external memory. If a database system is used for that purpose, the software requires at least several complex installation steps. Since Domain Model Lite uses XML files to save domain model data, there is no need for any special installation. In addition, Domain Model Lite allows the use of a database.

Domain Model RAD

Domain Model Lite has a companion rapid web application development framework, called Domain Model RAD, which can be used to make a default Wicket application out of a domain model. Domain Model RAD uses the domain model configuration to find the model entry points and to provide a web page for each entry point, either for the display or for the update of data. An entry point is a collection of entities and it is presented in a web page as a table, a list, or a slide show of entities. This choice and other view presentation properties are defined in the XML configuration of the domain model**.** The traversal of the domain model is done by navigating from an entry entity to neighbor entities following the parent-child neighbor directions.

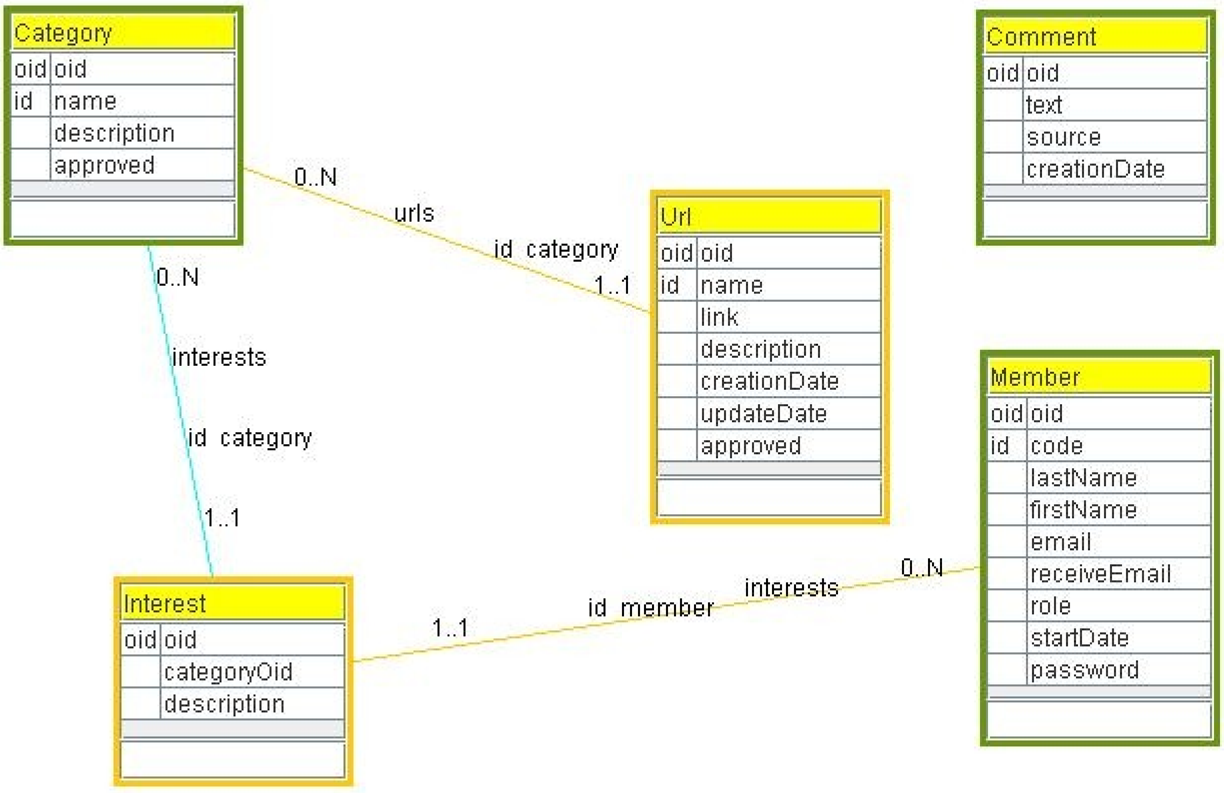
A default application may help developers validate and consequently refine the domain model. In addition, Domain Model RAD has a collection of web components that may be easily reused in specific web applications to display or update entities. For example, the component called `EntityUpdateTablePanel` is used to display entities as a table with a link to update the selected entity and a link to display child entities of the selected entity. The following is a list of the most important web components.

```
EntryUpdateTablePanel
EntityAddFormPanel
EntityUpdateConfirmRemovePanel
EntityEditFormPanel
EntryDisplayTablePanel
EntityDisplayTablePanel
EntityDisplayListPanel
EntityDisplaySlidePanel
EntityDisplayPanel
EntityDisplayMinPanel
EntityDisplayKeywordSelectionPanel
EntityDisplayLookupTablePanel
```

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 [8]. The following is a domain model of web links (or urls) that are of interest to certain members.

The Urls domain model concepts are: Url, Category, Member, Interest and Comment. Url describes a web link. Urls are categorized. Members express their interests in categories of web links.



A concept is described by its properties and neighbors. The Url concept has a name, a link, a description, a creation date, the last update date and whether it is approved or not. The Url concept has only one neighbor, the Category concept. However, the Category concept has two neighbors: the Url and Interest 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. The Category --> Url direction is named urls, its minimal cardinality is 0 and its maximal cardinality is N. Note that cardinalities are characteristics of the (source) concept. Thus, they are expressed close to the concept, which is opposite in UML. A Url has exactly one category.

A concept is represented in Domain Model Lite as two Java classes, one for `Entity` and the other for `Entities` (or `OrderedEntities` if an order of entities is important). The `Entity` class implements the `IEntity` interface, and the `Entities` class implements the `IEntities` interface.

```
public interface IEntity extends Serializable {
        public IDomainModel getDomainModel();
        public ConceptConfig getConceptConfig();
        public void setOid(Oid oid);
        public Oid getOid();
        public void setCode(String code);
        public String getCode();
        public Id getId();
        public void setProperty(String propertyCode, Object property);
        public Object getProperty(String propertyCode);
        public void setNeighborEntity(String neighborCode, IEntity
                    neighborEntity);
        public IEntity getNeighborEntity(String neighborCode);
        public void setNeighborEntities(String neighborCode,
                    IEntities neighborEntities);
        public IEntities getNeighborEntities(String neighborCode);
        public boolean update(IEntity entity) throws ActionException;
        public IEntity copy();
        public boolean equalContent(IEntity entity);
        public boolean equalOids(IEntity entity);
        public boolean equalIds(IEntity entity);
}

public interface IEntities extends Serializable {
        public IDomainModel getDomainModel();
        public ConceptConfig getConceptConfig();
        public Collection getCollection();
        public IEntities getEntities(SelectionCriteria selectionCriteria)
                    throws SelectionException;
        public IEntities getSourceEntities();
        public void setPropagateToSource(boolean propagate);
        public boolean isPropagateToSource();
        public IEntities union(IEntities entities) throws SelectionException;
        public IEntities intersection(IEntities entities) throws
                    SelectionException;
        public boolean isSubsetOf(IEntities entities) throws
                    SelectionException;
        public boolean add(IEntity entity) throws ActionException;
        public boolean remove(IEntity entity) throws ActionException;
        public boolean update(IEntity entity, IEntity updateEntity)
                    throws ActionException;
        public boolean contain(IEntity entity);
        public IEntity retrieveByOid(Oid oid);
        public IEntity retrieveByCode(String code);
        public IEntity retrieveByProperty(String propertyCode, Object
                    paramObject);
        public IEntity retrieveByNeighbor(String neighborCode, Object
                    paramObject);
        public Iterator iterator();
        public int size();
        public boolean isEmpty();
        public Errors getErrors();
}
```

The Category concept has two classes: `Category` and `Categories`. The Category concept is an entry point into the domain model. An entry point concept has a green border. Thus, the Url concept is not an entry point. That means that urls can be reached only through its category. However, an object of the `IEntities` type that represents all urls of all categories may be created by a specific method. The `Category` class describes the Category concept and the `Categories` class represents all categories. A specific category may have from 0 to N urls. The urls for that category, and other urls for other categories are represented by the `Urls` class. The `Url` class is used to describe the Url concept. In the context of the Category -- Url relationship, the Category concept is a parent, and the Url concept is a child.

Every concept in Domain Model Lite has two predefined properties: oid and code. The oid property is mandatory, while the code property is optional. The oid property is used as an artificial concept identifier and is completely managed by Domain Model Lite. Its value is a time stamp and it is unique universally**.** In addition, a concept may have, at most, one user oriented identifier (id) that consists of the concept 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. The Url concept id consists of the name property and the category neighbor (direction). Thus, a url name must be unique only within its category.

Entities as Plain Old Java Objects

The Category concept has two classes: `Category` and `Categories`. For the sake of space, only the Category concept and its relationship to the Url concept, will be considered in this example. The example is used to show the flavor of Domain Model Lite.

The `Category` class extends the `Entity` abstract class, which implements the `IEntity` interface. It contains three properties and one child neighbor with the corresponding `set` and `get` methods. All properties are typed by Java classes and not by Java base types. A `Boolean` property has also an `is` method that returns a `boolean` base value for convenience reasons. The class constructor passes the domain model to its inheritance parent. The neighbor is created in the class constructor. The neighbor `setUrls` method sets the current category as the neighbor parent.

```
public class Category extends Entity {

      private String name;
      private String description;
      private Boolean approved;

      private Urls urls;

      public Category(IDomainModel domainModel) {
            super(domainModel);
            urls = new Urls(this);
      }

      public void setName(String name) {
            this.name = name;
      }

      public String getName() {
            return name;
      }

      public void setDescription(String description) {
            this.description = description;
      }
```

```
	public String getDescription() {
		return description;
	}

	public void setApproved(Boolean approved) {
		this.approved = approved;
	}

	public Boolean getApproved() {
		return approved;
	}

	public boolean isApproved() {
		return getApproved().booleanValue();
	}

	public void setUrls(Urls urls) {
		this.urls = urls;
		urls.setCategory(this);
	}

	public Urls getUrls() {
		return urls.getUrlsOrderedByName();
	}
}
```

The `Categories` class extends the `OrderedEntities` abstract class, which in turn extends the abstract `Entities` class, which implements the `IEntities` interface. The class constructor passes the domain model to its inheritance parent. The `getCategory` is a convenience method that uses the Domain Model Lite `retrieveByProperty` method. The `getApprovedCategories` method shows how a selection of entities is done in Domain Model Lite. The `getCategoriesOrderedByName` method shows how entities are ordered. Both selection and order produce a new specific entities object.

```
public class Categories extends OrderedEntities {

	public Categories(IDomainModel domainModel) {
		super(domainModel);
	}

	public Category getCategory(String name) {
		return (Category) retrieveByProperty("name", name);
	}

	public Categories getApprovedCategories() {
		Categories approvedCategories = null;
		try {
			SelectionCriteria criteria =
			    SelectionCriteria.defineEqualCriteria(
			           "approved", Boolean.TRUE);
			approvedCategories = (Categories) getEntities(criteria);
		} catch (SelectionException e) {
			log.error("Error in Categories.getApprovedCategories: "
			           + e.getMessage());
		}
		return approvedCategories;
	}

	public Categories getCategoriesOrderedByName() {
```

```
            Categories orderByName = null;
            try {
                  CaseInsensitiveStringComparator cisc = new
                     CaseInsensitiveStringComparator();
                  PropertyComparator nameComparator = new
                      PropertyComparator("name", cisc);
                  OrderCriteria criteria =
                      OrderCriteria.defineOrder(nameComparator);
                  orderByName = (Categories) getEntities(criteria);
            } catch (OrderException e) {
                  log.error("
                      Error in Categories.getCategoriesOrderedByName: "
                      + e.getMessage());
            }
            return orderByName;
      }
}
```

Domain Model XML Configuration

A domain model must be configured in an XML configuration file [9]. This configuration reflects the model classes. However, it provides more information about the domain model default behavior used heavily in Domain Model RAD. The model XML configuration is loaded up-front and converted into Domain Model Lite meta entities. The following is a minimal version of the configuration.

```
<models>
  <model oid="2001">
    <code>Urls</code>
    <packagePrefix>org.urls</packagePrefix>
      <concepts>
        <concept oid="2020">
          <code>Category</code>
          <entityClass>org.urls.model.component.category.Category
          </entityClass>
          <entitiesCode>Categories</entitiesCode>
          <entitiesClass>org.urls.model.component.category.Categories
          </entitiesClass>
          <entry>true</entry>
          <fileName>categories.xml</fileName>
          <properties>
            <property oid="1">
              <code>name</code>
              <propertyClass>java.lang.String</propertyClass>
              <required>true</required>
              <unique>true</unique>
            </property>
            <property oid="2">
              <code>description</code>
              <propertyClass>java.lang.String</propertyClass>
            </property>
            <property oid="3">
              <code>approved</code>
              <propertyClass>java.lang.Boolean</propertyClass>
              <required>true</required>
              <defaultValue>false</defaultValue>
            </property>
          </properties>
          <neighbors>
```

```
            <neighbor oid="1">
              <code>urls</code>
              <destination>Url</destination>
              <internal>true</internal>
              <type>child</type>
              <min>0</min>
              <max>N</max>
            </neighbor>
          </neighbors>
        </concept>
        ...
    </concepts>
  </model>
</models>
```

Web Applications

I have developed two different web applications for the same Urls domain model. The first web application uses Wicket for the application views [10]. The second web application uses Domain Model RAD for the application views [11]. Thanks to Domain Model RAD, only a small portion of the application is specific, which means that there is significantly less Java code in the second application.

Conclusion

I have developed a domain model framework, called Domain Model Lite, to provide an easy support for small domain models, which are usually used in rapid web development. Besides, I have also developed a web component framework, called Domain Model RAD, which makes it possible to produce quickly a web application based on the given domain model. With Domain Model Lite, a web application developer does not need to learn a complex framework, such as Hibernate, in order to create and save a domain model. With Domain Model RAD, a web application developer may create a small application quickly. For example, I have developed the Public Point Presentation web application in three spirals: ppp00 [12], ppp01, and ppp02 [13]. The ppp00 spiral has been created with almost no programming [14], while the ppp02 spiral has only a few specific classes.

Web Links

[1] http://java-source.net/open-source/web-frameworks
[2] http://struts.apache.org/
[3] http://wicketframework.org/
[4] http://www.hibernate.org/
[5] http://drdb.fsa.ulaval.ca/dm/
[6] http://drdb.fsa.ulaval.ca/urls/
[7] http://www.dsmforum.org/
[8] http://drdb.fsa.ulaval.ca/mmLite/
[9] http://drdb.fsa.ulaval.ca/dm07/web/config/specific-model-config.xml
[10] http://drdb.fsa.ulaval.ca/urls08/code.html
[11] http://drdb.fsa.ulaval.ca/dm07/code.html
[12] http://drdb.fsa.ulaval.ca/ppp00/
[13] http://drdb.fsa.ulaval.ca/ppp02/
[14] http://drdb.fsa.ulaval.ca/ppp00/code.html