

SECTION II: Domain Model Views

Chapter 4: Model as a Web Application

The objective of this chapter is to make 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 by Modelibra and ModelibraWicket, with the help of the domain configuration. This web application can be considered a default application of the domain. With some small changes in the XML file of the domain 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, whose web pages will be designed once the model has been validated. This does not mean that the model will not change later on. However, model changes after its validation will slow down the development of the application.

For pedagogical reasons, the web application based on the WebLink model is not generated in this chapter. However, the TravelImpression model is used to show steps of the code generation. This model is about travel impressions of young travelers. The domain is Travel and the model is Impression. Thus, the project is called TravelImpression.

A default web application for the TravelImpression-01 project is created without any programming. First, the Eclipse project is prepared based on the TravelImpression-00 project. Second, the minimal domain configuration is generated from the model in ModelibraModeler to the config directory in the project. Third, the code is generated by the main method of the dm.gen.DmGenerator class in the project.

Domain Model

The WebLink model has one more concept: Comment. There are three specific properties: text, source and creationDate. A text provides a comment about something. A source may be a person that made a comment or a resource from which the comment was taken. A creation date indicates a date when the comment was introduced. For the time being, there is no relationship between the Url and Comment concepts.

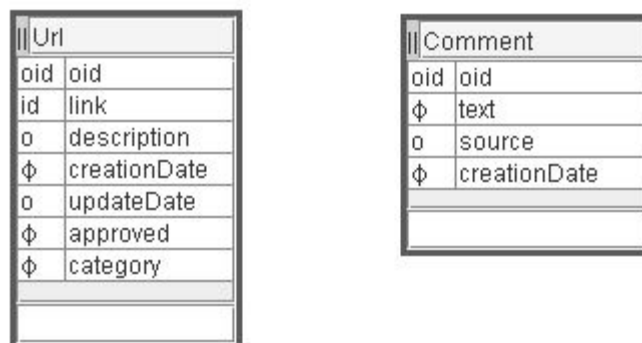


Figure 4.1. Url and Comment concepts

Web Application

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 application is executed by a web server. A web server listens to user requests that come from web browsers and delegates those requests to a corresponding web application. A delegated user request is used by the web application as the starting point to generate a response web page that is then passed to the web server. The server will send the response page to the user that has requested it.

There are three new technologies used in this chapter. A web server, called Jetty [Jetty], is used to execute a web application directly from Eclipse on a personal computer. A web framework, called Wicket [Wicket], is used by ModelibraWicket to compose a web application from the domain model. Both Jetty and Wicket are based on servlets [Servlet], the Java basic technology for the Web. A servlet is a Java class that knows the HTTP protocol for communicating on the Web.

Jetty is a fully featured web server for static and dynamic content. The web server and a web application run in the same process, without interconnection overheads and complications. Furthermore, as a pure Java component, Jetty can be simply included in a web application as a single jar file. The jar file is located in the lib directory. In this way, Jetty is included in each spiral and is used only for testing purposes.

Recently we have seen the proliferation of web frameworks whose main objective is to simplify the development of complex web applications. One such framework, Wicket, brings plain object oriented programming to web applications. 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.

Project Directories

In this chapter, a new project directory structure is introduced to support requirements for a dynamic web application. The application root directory name determines the web application name. In the chapter spiral, the application name is DmEduc-02. A web application must have the WEB-INF directory. However, this directory must be a direct sub-directory of the application root directory. This directory is well protected by the web server so that users cannot access directly the directory content. For that reason, the more sensitive information such as configurations and data are now in the WEB-INF directory.

```
DmEduc-02
  src
  test
  css
  doc
  mm
```

```
WEB-INF
  classes
  config
  data
  lib
  logs
```

There are several jar files in the lib directory required by Jetty, Wicket and Java servlets. In addition to the Modelibra jar file, which represents the domain model, there is also the ModelibraWicket jar file, which makes a web application out of the domain model.

The css directory contains the CSS files [CSS] that contain layout definitions for web pages.

Web Configuration

The WEB-INF directory contains the mandatory web application configuration file that must be called web.xml. Basically, the file is used to configure the application filter [Filter] that will be responsible for a communication between application users and the application servlet. The filter is in the form of a Java class from the Wicket framework: org.apache.wicket.protocol.http.WicketFilter. The filter class has a parameter that declares the web application class: dmeduc.wicket.app.DmEducApp.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE web-app
  PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
  "http://java.sun.com/dtd/web-app_2_3.dtd">

<web-app>

  <display-name>Modelibra Wicket Application</display-name>

  <filter>
    <filter-name>ModelibraWicketApplication</filter-name>
    <filter-class>org.apache.wicket.protocol.http.WicketFilter</filter-class>
    <init-param>
      <param-name>applicationClassName</param-name>
      <param-value>dmeduc.wicket.app.DmEducApp</param-value>
    </init-param>
  </filter>

  <filter-mapping>
    <filter-name>ModelibraWicketApplication</filter-name>
    <url-pattern>/app/*</url-pattern>
  </filter-mapping>

  <welcome-file-list>
    <welcome-file>index.html</welcome-file>
  </welcome-file-list>

</web-app>
```

The filter mapping makes a link between a user request pattern and the servlet responsible for providing the request response. When the Jetty web server is started locally, on a personal computer, the server web address is: <http://localhost:8081/>. The port number, where the server listens for user requests, is 8081. This is determined by the Jetty XML configuration file that can be found in the config directory. Since there is only one web application - /ModelibraWicket - configured as the active application in the Jetty configuration file, the application web address is: <http://localhost:8081/ModelibraWicket/>. When this address is used in a user request, Jetty finds the application and reads its web.xml configuration. Based on the configuration, Jetty tries to display the index.html welcome file, which is located in the application root directory.

```
<html>
<head>
  <meta http-equiv="Refresh" content="0; url=app" />
</head>
</html>
```

This file is used to forward the user request to the app URL (nothing to do with the Url concept in the WebLink model), which produces the new request: <http://localhost:8081/ModelibraWicket/app/>. This request is then mapped to the application filter based on the filter mapping, which states that any request to the current application that ends with /app/*, where * stands for any text, will be handled by the ModelibraWicketApplication filter, which is actually the org.apache.wicket.protocol.http.WicketFilter Wicket class. The filter knows that application class is dmeduc.wicket.app.DmEducApp. The DmEducApp class extends the DomainApp class (org.modelibra.wicket.app.DomainApp) that knows how to display the application home page.

Jetty Server

There is one Java class, called Start in the dmeduc.wicket.app package, which is used to start Jetty in Eclipse. The class is selected and executed as Java application, by using the class pop-up menu and the *Run As/Java Application* menu item. The following will be displayed in the Eclipse *Console* page:

```
11:24:10.609 EVENT Statistics on = false for org.mortbay.jetty.Server@1ad086a
11:24:10.625 EVENT Starting Jetty/4.2.22
11:24:10.640 EVENT Checking Resource aliases
11:24:10.812 EVENT Started WebApplicationContext[/ModelibraWicket,Modelibra Wicket Application]
11:24:13.234 WARN!! Delete existing temp dir C:\Temp\Jetty_8081_ModelibraWicket for
WebApplicationContext[/ModelibraWicket,Modelibra Wicket Application]
*****
*** WARNING: Wicket is running in DEVELOPMENT mode. ***
***               ^^^^^^^^^^ ***
*** Do NOT deploy to your live server(s) without changing this. ***
*** See Application#getConfigurationType() for more information. ***
*****
11:24:14.125 EVENT Started SocketListener on 0.0.0.0:8081
11:24:14.125 EVENT Started org.mortbay.jetty.Server@1ad086a
11:24:14.593 EVENT Started HttpContext[/]
```

The keyword to look for is *Started*. Now, we can use a web browser (for example, Firefox) to display the application main page: <http://localhost:8081/ModelibraWicket/>.

It is important to stop (terminate) the Jetty server at the end of testing (testing by using the web application), by clicking on the small red square in the Eclipse *Console* page. If you forget to do that and you start again Jetty the following will be displayed in the Eclipse *Console* page:

```
11:32:24.062 EVENT Statistics on = false for org.mortbay.jetty.Server@1ad086a
11:32:24.093 EVENT Starting Jetty/4.2.22
11:32:24.093 EVENT Checking Resource aliases
11:32:24.265 EVENT Started WebApplicationContext[/ModelibraWicket,Modelibra Wicket Application]
11:32:25.437 WARN!! Delete existing temp dir C:\Temp\Jetty_8081_ModelibraWicket for
WebApplicationContext[/ModelibraWicket,Modelibra Wicket Application]
*****
*** WARNING: Wicket is running in DEVELOPMENT mode. ***
*** ^^^^^^^^^^ ***
*** Do NOT deploy to your live server(s) without changing this. ***
*** See Application#getConfigurationType() for more information. ***
*****
11:32:25.796 WARN!! Failed to start: SocketListener@0.0.0.0:8081
11:32:25.875 EVENT Stopped WebApplicationContext[/ModelibraWicket,Modelibra Wicket Application]
11:32:25.875 EVENT Stopped org.mortbay.jetty.Server@1ad086a
```

This time the keyword is *Stopped*. When this happens you will have to click on the XX icon in the Eclipse *Console* page (*Remove All Terminated Launches*) to be able to stop the Jetty server, by clicking on the small red square in the Eclipse *Console* page.

DmEduc Application

The home page 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 appears in the the title section.

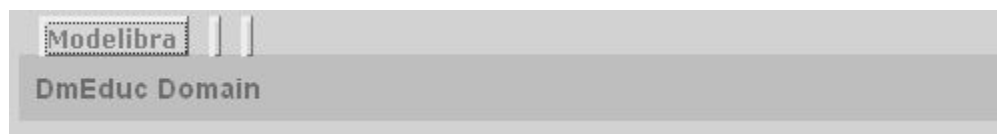


Figure 4.2 Home page

The *Modelibra* page is the application's domain page with a table of domain models. The domain title is *Modelibra – Education 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.

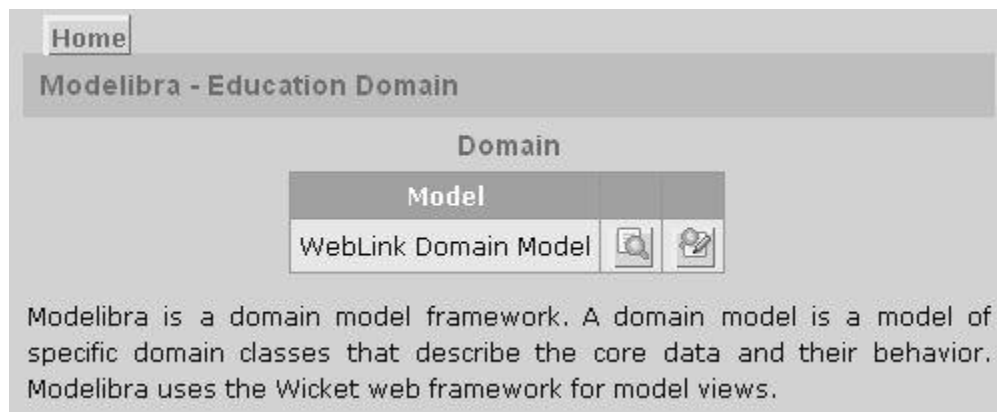


Figure 4.3. Domain page

The domain page has also an image of the meta model of Modelibra with five concepts: Domain, DomainModel, Concept, Property and Neighbor. The meta model is hierarchical and its only entry point is the Domain concept. A domain may have several models. A model may have many concepts. A concept may have many properties and neighbors.

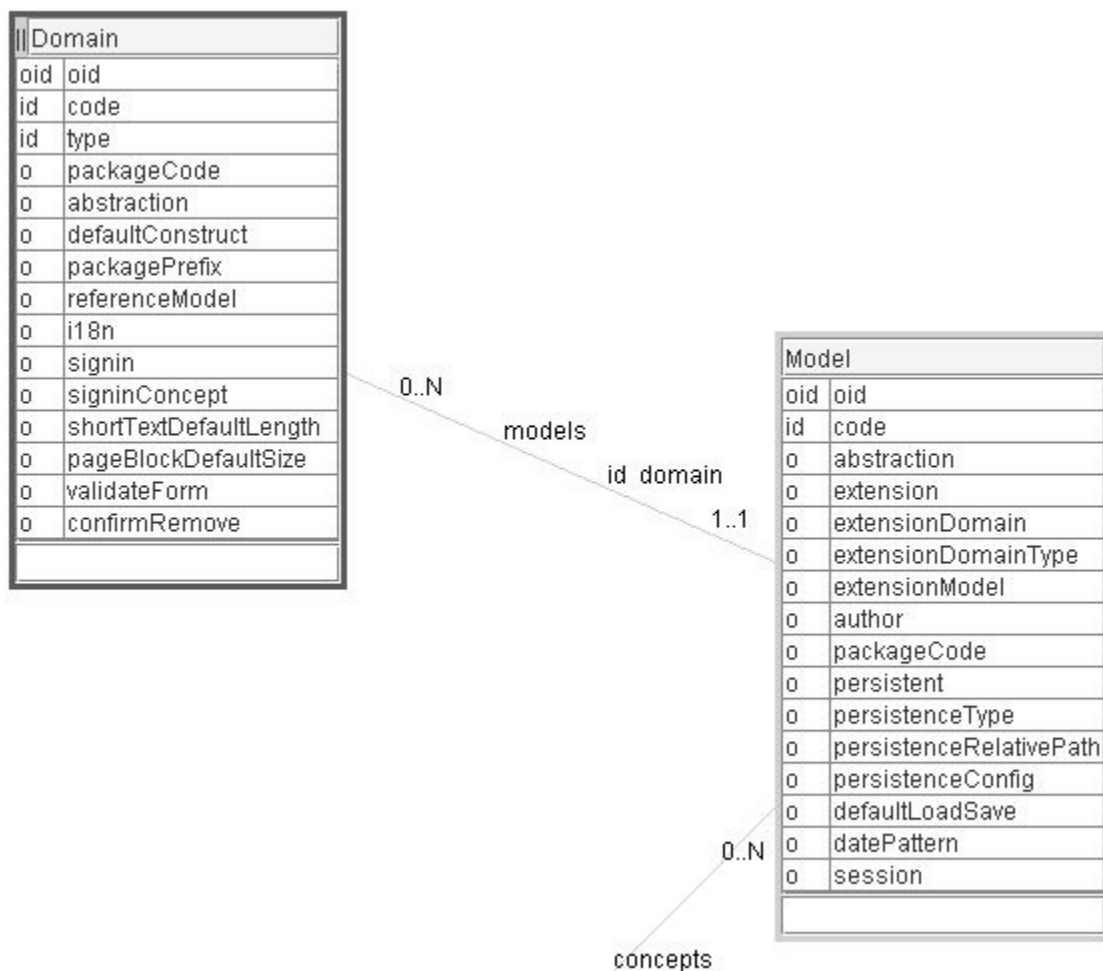


Figure 4.4.a. Modelibra meta model: Domain and DomainModel concepts

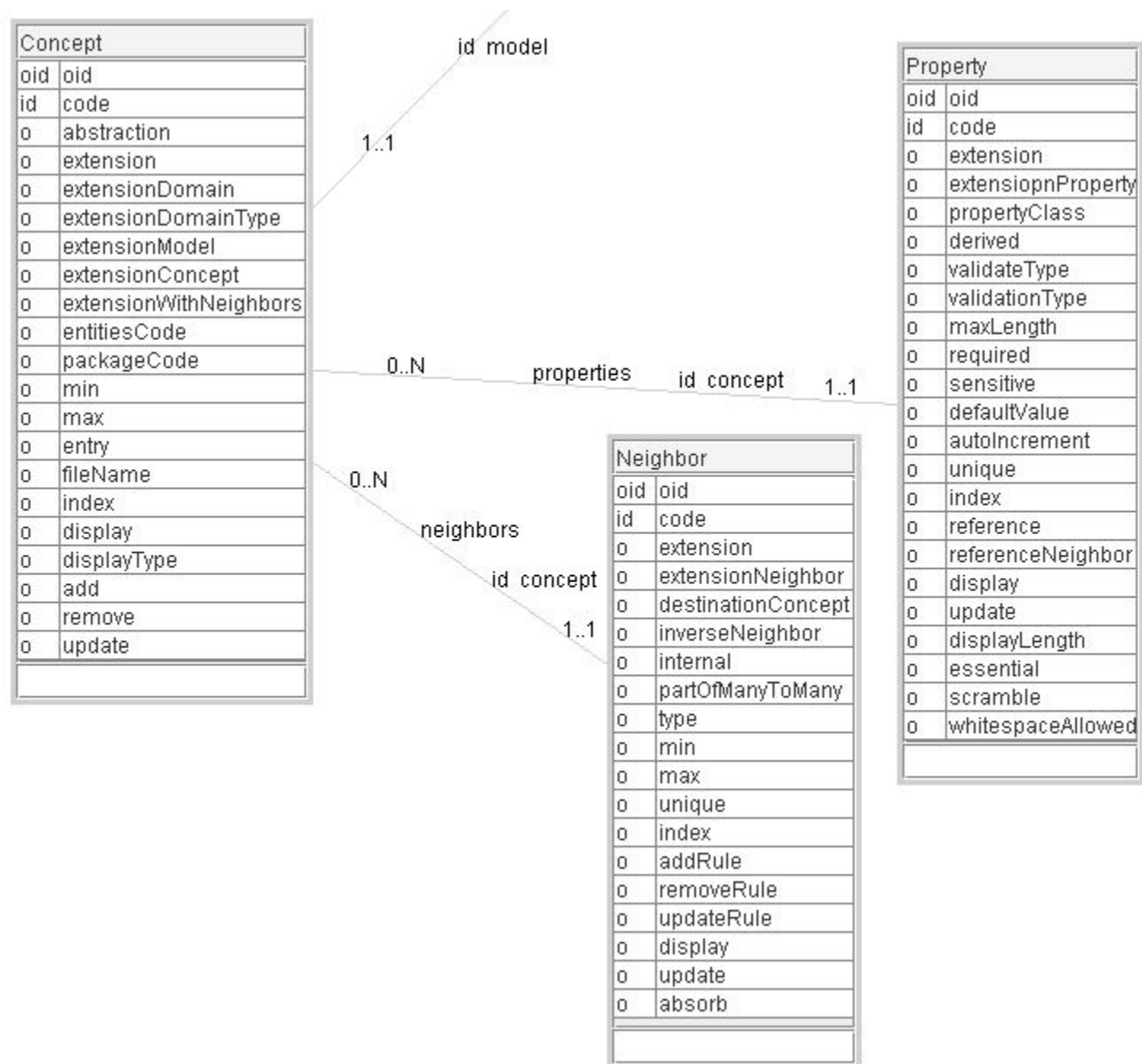


Figure 4.4.b. Modelibra meta model: Concept, Property and Neighbor concepts

The display model page presents a table of model entries. There are two entry concepts: *Comment* and *Url*. For each concept there are *Display* and *Select* links. The *Display* link presents the concept as a table of entities, with only required properties displayed. Each entity may be further displayed with all its details. The *Select* link provides a generic selection based on the selected property.

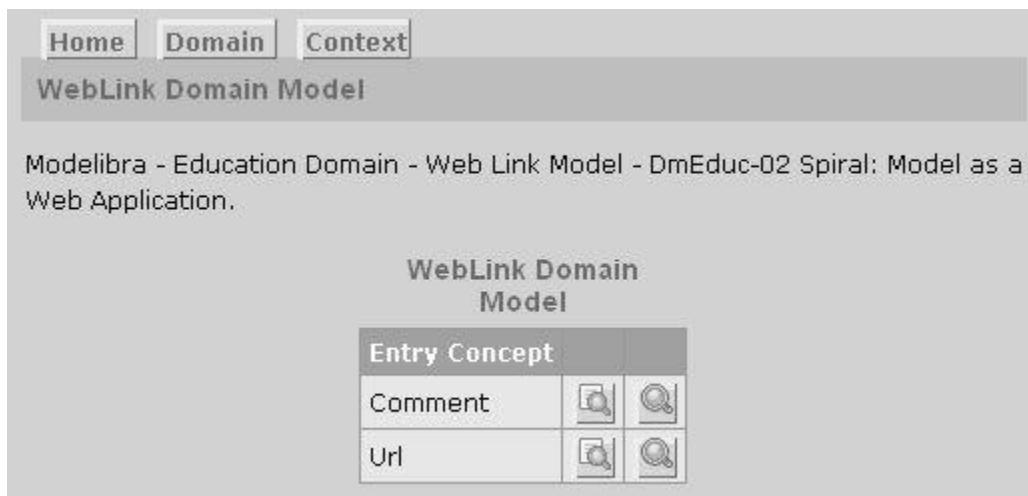


Figure 4.5. Entry concepts display

The update model page presents a table of model entries with *Update* links.

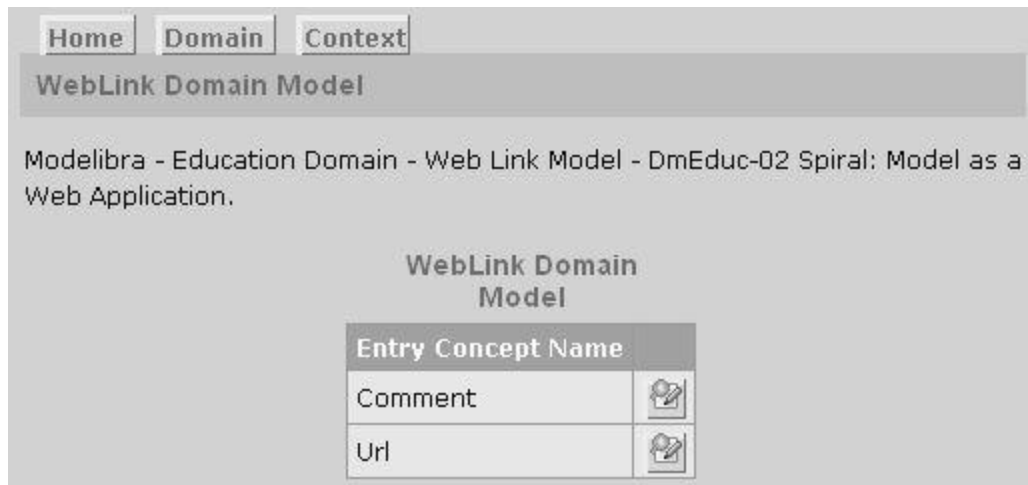


Figure 4.6. Entry concepts update

The entry concept has its own display or 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. In Figure 4.7. comments are displayed with only two properties.





Home	Domain	Context
WebLink Domain Model		
Comments		
Text	Creation Date	
This is something!	2007-02-02	
What is the magic?	2007-02-02	
There is no magic. It is all in the configuratio...	2007-02-07	
The new name for dmLite is Modelibra. The new na...	2007-09-04	
<< < 1 > >>		

Figure 4.7 Display table of comments

An entity may be displayed, selected or updated. For example, a new comment can be added and an existing comment can be displayed with all details, edited or removed.













Home

Domain

Context

WebLink Domain Model

Comments

Text	Creation Date			
This is something!	2007-02-02			
What is the magic?	2007-02-02			
There is no magic. It is all in the configuratio...	2007-02-07			
The new name for dmLite is Modelibra. The new na...	2007-09-04			

<< < 1 > >>




Figure 4.8. 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 4.9, an empty url is added. When the add form is displayed, the *Save* button is clicked to add an empty url. Since both a web link and a category of the web link are required properties, Wicket displays the two corresponding errors in front of the form, and a small star after a field in question shows that there is a problem with the value of the field.

HomeDomainContext

WebLink Domain Model

Field 'Web Link' is required.
Field 'Category' is required.

Url

Web Link	<input type="text"/> *
Description	<div></div>
Creation Date	<input type="text" value="2007-09-24"/>
Update Date	<input type="text"/>
Approved	<input type="checkbox"/>
Category	<input type="text"/> *
Save <input type="button" value="+"/> Cancel <input type="button" value="Φ"/>	

Figure 4.9. Add an empty url

Home Domain Context

WebLink Domain Model

Web Link must be well-formed.

Url

Web Link	????
Description	
Creation Date	2007-09-24
Update Date	
Approved	<input type="checkbox"/>
Category	Personal
Save <input type="button" value="+"/> Cancel <input type="button" value="Φ"/>	

Figure 4.10. Add a not valid url

In Figure 4.10, a not valid url is refused with the message

Web Link must be well-formed.

Specific messages related to the Url concept can be found in the DmEducApp.properties file:

```

Url=Url
Urls=Urls
Url.link=Web Link
Url.description=Description
Url.creationDate=Creation Date
Url.updateDate=Update Date
Url.approved=Approved
Url.category=Category
Url.id=Web Link
Url.id.unique=Web Link must be unique.
Url.link.length=Web Link is too long.
Url.link.required=Web Link is required.
Url.link.validation=Web Link must be well-formed.
Url.description.length=Description is too long.
Url.creationDate.length=Creation Date is too long.

```

Url.creationDate.required=Creation Date is required.
Url.updateDate.length=Update Date is too long.
Url.approved.required=Approved is required.
Url.category.length=Category is too long.

Domain Configuration

The specific-domain-config.xml file has been modified to include the configuration for the Comment concept.

```
<concept oid="110110120">
  <code>Comment</code>
  <entry>true</entry>

  <properties>
    <property oid="110110120110">
      <code>text</code>
      <propertyClass>java. lang. String</propertyClass>
      <maxLength>1020</maxLength>
      <required>true</required>
    </property>
    <property oid="110110120120">
      <code>source</code>
      <propertyClass>java. lang. String</propertyClass>
      <maxLength>255</maxLength>
    </property>
    <property oid="110110120130">
      <code>creationDate</code>
      <propertyClass>java. util. Date</propertyClass>
      <maxLength>16</maxLength>
      <required>true</required>
      <defaultValue>today</defaultValue>
    </property>
  </properties>
</concept>
```

Concept Classes

There are two new methods in the Urls class. The getUrls method selects a subset of entities based on the chosen property. The postUpdate method is used after the update action is done to provide the update date.

```
public Urls getUrls(String propertyCode, Object property) {
    return (Urls) selectByProperty(propertyCode, property);
}

protected boolean postUpdate(Url beforeUrl, Url afterUrl) {
```

```

        if (super.postUpdate(beforeUrl, afterUrl)) {
            Date today = new Date();
            Date updateDate = afterUrl.getUpdateDate();
            EasyDate easyToday = new EasyDate(today);
            EasyDate easyUpdateDate = new EasyDate(updateDate);
            if (!easyToday.equals(easyUpdateDate)) {
                Url afterAfterEntity = afterUrl.copy();
                afterAfterEntity.setUpdateDate(today);
                return update(afterUrl, afterAfterEntity);
            } else {
                return true;
            }
        }
        return false;
    }
}

```

The postUpdate method, with two arguments, overrides the corresponding method from the Entities class. This specific postUpdate method is called by Modelibra. Its objective is to set the update date of the url entity. First, the postUpdate method from the inheritance parent is called to do whatever Modelibra does in the post update action, and, if it returns true, the post update trigger action continues. The EasyDate class from Modelibra is used to transform a Date object that has both date and time values into an EasyDate object that has only date value.

```
import org.modelibra.type.EasyDate;
```

With the EasyDate class, the comparison of two dates is straightforward. With the Date class, two dates may be of the same date but be still different since the date times are most probably different.

Modelibra has the pre and post methods for the add, remove and update methods of the Entities class. The pre and post protected methods may be used as actions to trigger some specific operations or as additional semantic integrity constraints. For example, the framework uses the preAdd and preUpdate methods to validate concepts and their properties by using XML configurations. Whatever Modelibra does not support, a programmer may add in specific pre and post methods.

There are two new classes for the Comment concept in the dmeduc.weblink.comment package: Comment and Comments. There is nothing new in those two classes.

The WebLink model class has been modified to include the Comments entry.

Concept Tests

Comment is the new concept in the model. Thus, there is a new test class for the Comment concept. The CommentTest class can be found in the test source code directory in the dmeduc.weblink.comment package. There are two tests and they both pass without errors or failures.

Before and after test actions are the same as in the tests of the Url concept. The only difference is that tests are done using the collection of Comment entities.

```

private static Comments comments;

@BeforeClass
public static void beforeTests() throws Exception {
    comments = DmEducTest.getSingleton().getDmEduc().getWebLink().getComments();
}

@Before
public void beforeTest() throws Exception {
    comments.getErrors().empty();
}

@After
public void afterTest() throws Exception {
    for (Comment comment : comments.getList()) {
        comments.remove(comment);
    }
}

@AfterClass
public static void afterTests() throws Exception {
    DmEducTest.getSingleton().close();
}

```

The first test creates four valid comments with required values.

```

@Test
public void commentsRequiredCreated() throws Exception {
    Comment comment01 = comments.createComment("Modelibra is magic.");
    Comment comment02 = comments
        .createComment("Modelibra is a domain model framework.");
    Comment comment03 = comments
        .createComment("Wicket is a web framework.");
    Comment comment04 = comments
        .createComment("Wicket is a small gate or door.");

    assertNotNull(comment01);
    assertTrue(comments.contain(comment01));
    assertNotNull(comment02);
    assertTrue(comments.contain(comment02));
    assertNotNull(comment03);
    assertTrue(comments.contain(comment03));
    assertNotNull(comment04);
    assertTrue(comments.contain(comment04));
    assertTrue(comments.getErrors().isEmpty());
}

```

The second test refuses the creation of the null comment.

```

@Test

```

```

public void textRequired() throws Exception {
    Comment comment = comments.createComment(null);

    assertNull(comment);
    assertFalse(comments.contains(comment));
    assertFalse(comments.getErrors().isEmpty());
    assertNotNull(comments.getErrors().getError("Comment. text. required"));
}

```

Now, all tests in this project can be executed by selecting the project and *Run As/JUnit Tests* of the context menu.

Application View

Given a domain model, there is no much work to make the model alive with ModelibraWicket. There is only one specific package: dmeduc.wicket.app. There are three files in the corresponding directory. Start.java does not change from spiral to spiral. It is used to start the Jetty server within Eclipse. The DmEducApp.java and DmEducApp.properties represent the only specific work related to application views done in this spiral. The application class has the application domain property. Its constructor provides the domain configuration to the domain class. The domain is constructed and then passed to the persistent domain class. However, only the domain property is used in the application programming.

```

package dmeduc.wicket;

import org.modelibra.wicket.app.DomainApp;

import dmeduc.DmEduc;
import dmeduc.DmEducConfig;
import dmeduc.PersistentDmEduc;

public class DmEducApp extends DomainApp {

    private DmEduc dmEduc;

    public DmEducApp() {
        super();
        DmEducConfig dmEducConfig = new DmEducConfig();
        dmEduc = new DmEduc(dmEducConfig.getDomainConfig());
        setPersistentDomain(new PersistentDmEduc(dmEduc));
    }

    public DmEduc getDmEduc() {
        return dmEduc;
    }

}

```

The DmEducApp.properties file has information and error messages. The error keys, in the dot form,

are required by Modelibra and the messages are the only specific items in this file. The messages are related to the DmEduc domain, the WebLink model and the Comment and Url concepts.

DmEduc=DmEduc Domain

DmEduc.title=Modelibra – Education Domain

DmEduc.description=Modelibra – Education Domain: Example models.

WebLink=WebLink Domain Model

WebLink.title=Modelibra – Education Domain – Web Link Model

WebLink.description=Modelibra – Education Domain – Web Link Model – DmEduc-02 Spiral: Model as a Web Application.

Comment=Comment

Comments=Comments

Comment.text=Text

Comment.source=Source

Comment.creationDate=Creation Date

Comment.text.length=Text is too long.

Comment.text.required=Text is required.

Comment.source.length=Source is too long.

Comment.creationDate.length=Creation Date is too long.

Comment.creationDate.required=Creation Date is required.

Url=Url

Urls=Urls

Url.link=Web Link

Url.description=Description

Url.creationDate=Creation Date

Url.updateDate=Update Date

Url.approved=Approved

Url.category=Category

Url.id=Web Link

Url.id.unique=Web Link must be unique.

Url.link.length=Web Link is too long.

Url.link.required=Web Link is required.

Url.link.validation=Web Link must be well-formed.

Url.description.length=Description is too long.

Url.creationDate.length=Creation Date is too long.

Url.creationDate.required=Creation Date is required.

Url.updateDate.length=Update Date is too long.

Url.approved.required=Approved is required.

Url.category.length=Category is too long.

Modelibra Interfaces

In Modelibra, the Entity class implements the IEntity interface and the Entities class implements the IEntities interface. In the DmEduc-02 spiral there is only one new method used from the two interfaces. The method is selectByProperty from the IEntities interface. The method selects entities whose property, determined by the property code, is equal to the property object. If there are no qualified

entities, the returned collection is empty.

```
public interface IEntity<T extends IEntity> extends Serializable, Comparable<T> {

    public T copy();

}

public interface IEntities<T extends IEntity> extends Serializable, Iterable<T> {

    public IDomainModel getModel();

    public boolean add(T entity);

    public boolean update(T entity, T updateEntity);

    public T retrieveByOid(Oid oid);

    public T retrieveByProperty(String propertyCode, Object property);

    public Errors getErrors();

    public IEntities<T> selectByProperty(String propertyCode, Object property);

}
```

Travel Impression

A default web application for the Travel Impression model is created in this chapter without any programming (TravelImpression-01). The main objective of a default application is to validate the domain model, so that changes to the model are minimized in the later spirals of the project. In addition, a default application serves as a reasonable starting point for developing a more advanced spiral.

The domain of the project is Travel. The principal model of the Travel domain is Impression. In a nutshell, the idea is to create a web application that will allow young travelers to inform their families and friends about their impressions on the trip without losing too much time. With the help from someone in a family or from a friend, their impressions of visited places may be enriched by web links and photos. Of course, the traveler can do all of that without help from other people.

The first step is to choose a model that will be used to generate a domain configuration. The model in Figure 4.11 is used for that purpose. The model is designed with ModelibraModeler and the domain configuration will be generated by ModelibraModeler. It is important to verify that the model is complete with respect to Modelibra requirements.

All properties of all concepts must be typed. A model is semantically divided into hierarchies of sub-models, where each sub-model starts with an entry concept. Starting with an entry concept and following one-to-many internal neighbors determines the scope of a sub-model. Sub-models are related

by external neighbors. There are three hierarchical sub-models: Traveler (with Message, Photo and Note), Country and Place. The Note concept represents a many-to-many relationship between the Message and Place concepts (the X sign). Concept plural names are verified in the *Dictionary/Boxes...* menu item of the *Diagram* window of ModelibraModeler.

The model is not the last model spiral model from the previous chapter. When creating a model in spirals, we may create more spirals to get a better understanding of the model domain. However, we must be reasonable in selecting a not too complex model spiral as the starting point for programming spirals.

The chosen model spiral in Figure 4.11 is taken out of the model with multiple spirals by selecting all concepts and relationships in the *Diagram* window with the *Edit/Select All* menu item. Then, the selection is exported by the *Transfer/Export Selection...* menu item as the *TravelImpression.diagram* file. Property types are exported in the *Main* window by the *Transfer/Export Types...* menu item as the *ModelibraModeler.type* file. Those two files will be used to create the new model that will be saved in the *TravelImpression.mm* file.

We have to prepare now a new Eclipse project where the new model will be saved and from which the domain configuration will be generated. The new project is prepared from the *TravelImpression-00* project or from the *ModelibraWicketSkeleton* project. A way to do it is to export the source project without the *css*, *template*, *classes*, *lib* and *logs* directories to a new location. This is done in Eclipse by using *File/Export.../General/File System*. We will see that there are no SVN directories and files in the exported project. The *.project* file is then opened with a text editor and the project name is changed to *TravelImpression-01*. Finally, the new project is imported into Eclipse by *File/Import.../General/Existing Projects into a Workspace*.

The new *TravelImpression-01* project is then transferred to a Subversion (SVN) repository. This is done by selecting the project and using the *Team/Share Project...* pop-up menu item. If there is no need to share the project with other people, the initial step of exporting the source project should have included the *css*, *template*, *classes*, *lib* and *logs* directories.

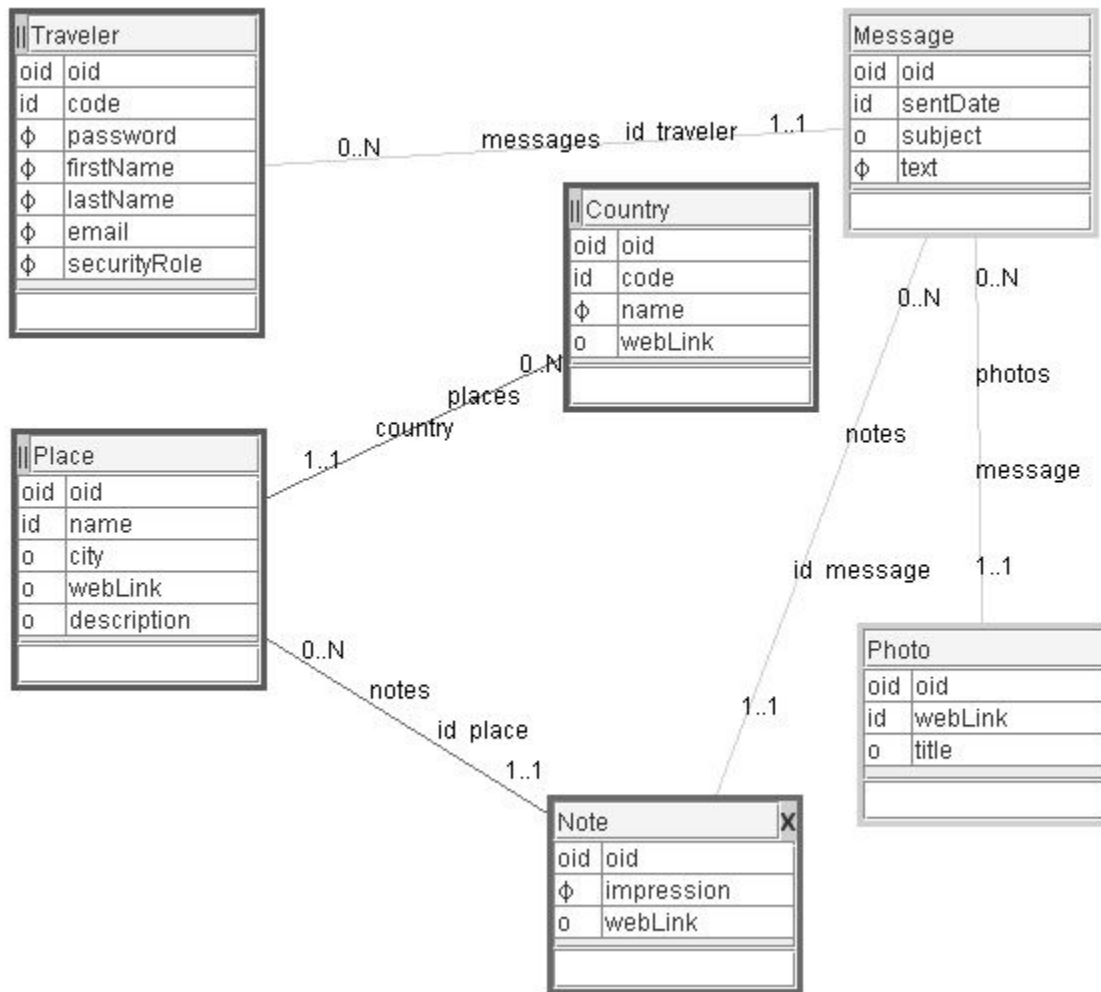


Figure 4.11. Travel Impression model

In the first transfer of files to a SVN repository, the empty classes directory, if the classes directory is in the list of directories and files to transfer, is excluded from the transfer by unchecking the check box for the classes directory. The `svn:ignore` property is defined at the WEB-INF directory for the classes and logs directories. This is done by selecting the WEB-INF directory and using the *Team/Set Property...* pop-up menu item. As a consequence, both the classes and logs directories will not be transferred to the repository in future commits. In addition, the `svn:externals` property is defined at the WEB-INF directory for the lib directory.

lib <http://.../Modelibra/trunk/ModelibraWicketSkeleton/WEB-INF/lib>

Those two properties are committed to the repository by the *Team/Commit...* pop-up menu item. After that, the update of the local project version (*Team/Update*) by the repository version loads the library files to the lib directory from the ModelibraWicketSkeleton project. In future, if the libraries in the ModelibraWicketSkeleton project change, the corresponding libraries in the local project will be updated whenever the update of the local project version is done.

Similarly, the `svn:externals` property is defined at the project root directory for the css and template directories.

css <http://.../Modelibra/trunk/ModelibraWicketCss/css>
template <http://.../Modelibra/trunk/ModelibraWicketSkeleton/template>

The local changes are committed to the repository by *Team/Commit...* The local version is then updated by *Team/Update* and the css and template directories with their content are loaded from the ModelibraWicketSkeleton project. In this way the local project will be in sync with future changes of the ModelibraWicketSkeleton project. Of course, if those changes are not wanted, the svn:externals property is not used, and the copy of corresponding directories and files is done once at the creation of the local project.

The ModelibraModeler.type and TravelImpression.mm files are placed in the mm directory of the project. When the project is selected, it may be refreshed by *File/Refresh*. The Travel Impression model is opened with ModelibraModeler. In the *Diagram* window, the *Generation/Generate XML Minimal Configuration of the Model...* menu item is used to generate the model configuration in the reusable-domain.config.xml file, which is located in the WEB-INF/config directory of the project. After the project is refreshed, the configuration file may be opened to examine the generated content. If there are no problems, the programming code may be generated by using the XML configuration file. The generated Java code is either generic and in that case the Java class name starts with the Gen prefix, or specific. The generic classes should not be changed by a programmer. Only the specific classes beg for a change. In this way, the code can be regenerated without losing the specific changes.

In the src directory there is the dm.gen package with several Java classes. The DmGenerator class must be slightly edited to introduce the domain code, which is in our case Travel, and the domain type, which is Reusable. For pedagogical reasons, the presented code may be simpler than what can be found in the ModelibraWicketSkeleton project.

```
package dm.gen;

import org.modelibra.config.DomainConfig;

public class DmGenerator {

    private static final String DOMAIN_CODE = "Travel";

    private static final String DOMAIN_TYPE = "Reusable";

    private DmModelibraGenerator dmModelibraGenerator;

    private DmModelibraWicketGenerator dmModelibraWicketGenerator;

    public DmGenerator() {
        dmModelibraGenerator = new DmModelibraGenerator(DOMAIN_CODE, DOMAIN_TYPE);

        DomainConfig domainConfig = dmModelibraGenerator.getDomainConfig();
        String authors = dmModelibraGenerator.getAuthors();
        String codeDirectoryPath = dmModelibraGenerator
            .getCodeDirectoryPath();

        dmModelibraWicketGenerator = new DmModelibraWicketGenerator(
            domainConfig, authors, codeDirectoryPath);
    }
}
```

```

    }

    ...

    public static void main(String[] args) {
        DmGenerator dmGenerator = new DmGenerator();
        dmGenerator.getDmModelibraGenerator().generate();
        dmGenerator.getDmModelibraWicketGenerator().generate();
    }
}

```

The main method is executed by selecting the class and by using the pop-up menu and its *Run As/Java Application* menu item. The main method creates an object of the DmGenerator class and calls two generate methods, one of the DmModelibraGenerator class, then another of the DmModelibraWicketGenerator class.

What is generated by the Modelibra generator can be deduced from the generateModelibraPartially method. This method is equivalent to the generate method. However, it allows a partial generation or regeneration of the code by commenting out unneeded method calls.

```

package dm.gen;

import org.modelibra.config.ConceptConfig;
import org.modelibra.config.DomainConfig;
import org.modelibra.config.ModelConfig;
import org.modelibra.gen.DomainGenerator;
import org.modelibra.gen.DomainModelGenerator;
import org.modelibra.gen.ModelibraGenerator;

public class DmModelibraGenerator {

    private DomainConfig domainConfig;

    private ModelibraGenerator modelibraGenerator;

    private String authors;

    private String sourceDirectoryPath;

    private String testDirectoryPath;

    public DmModelibraGenerator(String domainCode, String domainType) {
        DmConfig dmConfig = new DmConfig(domainCode, domainType);
        domainConfig = dmConfig.getDomainConfig();
        modelibraGenerator = new ModelibraGenerator(domainConfig);

        authors = modelibraGenerator.getAuthors();
        sourceDirectoryPath = modelibraGenerator.getSourceDirectoryPath();
        testDirectoryPath = modelibraGenerator.getTestDirectoryPath();
    }
}

```

```

...

public void generate() {
    modelibraGenerator.generate();
}

public void generateModelibraPartially() {
    DomainGenerator domainGenerator = modelibraGenerator
        .getDomainGenerator();

    domainGenerator.generateDomainConfig();
    domainGenerator.generateGenDomain();
    domainGenerator.generateDomain();
    domainGenerator.generateDomainTest();
    domainGenerator.generatePersistentDomain();

    for (ModelConfig modelConfig : domainConfig.getModelsConfig()) {
        DomainModelGenerator modelGenerator = new DomainModelGenerator(
            modelConfig, sourceDirectoryPath, testDirectoryPath);
        modelGenerator.generateGenModel();
        modelGenerator.generateModel();
        modelGenerator.generateModelTest();

        for (ConceptConfig conceptConfig : modelConfig.getConceptsConfig()) {
            modelGenerator.generateGenEntity(conceptConfig);
            modelGenerator.generateGenEntities(conceptConfig);
            modelGenerator.generateEntity(conceptConfig);
            modelGenerator.generateEntities(conceptConfig);
        }

        modelGenerator.generateEmptyXmlDataFiles();
    }
}

...
}

```

Similarly, the DmModelibraWicketGenerator class has both generate and generateModelibraWicketPartially methods.

```

package dm.gen;

import org.modelibra.config.ConceptConfig;
import org.modelibra.config.DomainConfig;
import org.modelibra.config.ModelConfig;
import org.modelibra.wicket.gen.DomainWicketGenerator;
import org.modelibra.wicket.gen.ModelibraWicketGenerator;

public class DmModelibraWicketGenerator {

```

```

private ModelibraWicketGenerator modelibraWicketGenerator;

private DomainConfig domainConfig;

public DmModelibraWicketGenerator(DomainConfig domainConfig,
    String authors, String codeDirectoryPath) {
    this.domainConfig = domainConfig;
    modelibraWicketGenerator = new ModelibraWicketGenerator(
        domainConfig, authors, codeDirectoryPath);
}

public void generate() {
    modelibraWicketGenerator.generate();
}

public void generateModelibraWicketPartially() {
    DomainWicketGenerator domainWicketGenerator = modelibraWicketGenerator
        .getDomainWicketGenerator();

    domainWicketGenerator.generateWebXml();
    domainWicketGenerator.generateDomainAppProperties();
    domainWicketGenerator.generateStart();
    domainWicketGenerator.generateDomainApp();

    for (ModelConfig modelConfig : domainConfig.getModelConfigs()) {
        for (ConceptConfig conceptConfig : modelConfig.getConceptConfigs()) {
            domainWicketGenerator
                .generateDomainModelConceptUpdateTablePage(conceptConfig);
            domainWicketGenerator
                .generateDomainModelConceptDisplayTablePage(conceptConfig);
            domainWicketGenerator
                .generateDomainModelConceptDisplayListPage(conceptConfig);
            domainWicketGenerator
                .generateDomainModelConceptDisplaySlidePage(conceptConfig);
        }
    }
}
}

```

Refresh the project and examine what was generated by comparing the two partial generation methods and the generated files.

Certain Java files may have some importing and formatting problems. When the class is selected in the upper right section of Eclipse, use the *Ctrl-Shift O F S* keyboard keys to import, format and save the class. The *Ctrl* and *Shift* keys are pressed with the left hand, and the letters *O*, *F* and *S* are pressed, one after another, with the right hand.

The generated code may have some errors even after the previous operation. With the regenerated code

in subsequent chapters, corrections of those types of errors will be explained. In this spiral there is one problem. The `getNotes` method in the `GenPlace` class of the `travel.impression.place` package makes a reference to a non existing method in the `Impression` class of the `travel.impression` package.

```
public Notes getNotes() {
    if (notes == null) {
        Impression impression = (Impression) getModel();
        Messages messages = impression.getMessages();
        setNotes(messages.getPlaceNotes((Place) this));
    }
    return notes;
}
```

The `Message` concept is not an entry point into the `Impression` model and the `getMessages` method does not exist in the `GenImpression` class. After the following code is added to the `Impression` class, after the specific code comment, the error disappears. Methods of this nature may be also generated in future.

```
public Messages getMessages() {
    Messages allMessages = null;
    boolean dataLoaded = isInitialized();
    if (dataLoaded) {
        try {
            allMessages = new Messages(this);
            allMessages.setPersistent(false);
            allMessages.setPre(false);
            allMessages.setPost(false);
            Travelers travelers = getTravelers();
            for (Traveler traveler : travelers) {
                Messages travelerMessages = traveler.getMessages();
                for (Message message : travelerMessages) {
                    allMessages.add(message);
                }
            }
        } catch (Exception e) {
            /log.error("Error in Impression.getMessages: " + e.getMessage());
        } finally {
            allMessages.setPersistent(true);
            allMessages.setPre(true);
            allMessages.setPost(true);
        }
    }
    return allMessages;
}
```

The `getMessages` method derives a virtual entry point in such a way that all messages of all travelers are collected in the same collection of message entities. This is done only after the model is initialized by loading it from data files. Since this is an expensive operation, the new collection of all messages will not be saved, and all validations and propagations in the pre and post actions of the add, remove and update methods of the `IEntities` interface will not be done.

Again, it is important to note that there are classes that start with the `Gen` prefix. Those classes should

not be edited by programmers. All additions must be done in specific classes. In this way, even if a model changes, only the Gen classes may be regenerated and the specific code will stay intact.

You may use the `ImpressionTest` class in the `travel.impression` package to test the model and learn more about Modelibra.

After the code generation, all packages and classes generated by the following code from the `DmModelibraWicketGenerator` class are deleted within Eclipse. Those classes will be regenerated in a future spiral.

```
for (ModelConfig modelConfig : domainConfig.getModelsConfig()) {  
  
    for (ConceptConfig conceptConfig : modelConfig.getConceptsConfig()) {  
        domainWicketGenerator  
            .generateDomainModelConceptUpdateTablePage(conceptConfig);  
        domainWicketGenerator  
            .generateDomainModelConceptDisplayTablePage(conceptConfig);  
        domainWicketGenerator  
            .generateDomainModelConceptDisplayListPage(conceptConfig);  
        domainWicketGenerator  
            .generateDomainModelConceptDisplaySlidePage(conceptConfig);  
    }  
}
```

The template directory contains the code templates used by Modelibra to generate the code. The templates are not that easy to read. If you want some other code to be generated by Modelibra, please make a requirement request at JavaForge [JavaForge] in the Modelibra project.

The content of the `css` directory may be updated to adapt CSS [CSS] declarations to your needs. However, you should not delete or rename existing `.css` files. You may add your own CSS files that you want to use in your specific web pages. However, if you want to have a completely different way of presenting web pages, do not use the `svn:externals` property at the project's root directory for the `css` directory. Copy the `cvs` directory initially from the `ModelibraWicketSkeleton` project and change definitions of the CVS classes but do not rename the class names since they are used by `ModelibraWicket`.

The data files are generated as XML empty documents. If you add some new data by running the web application, it is your responsibility to backup the data files. Without the backup files, if you regenerate them, their content will be gone.

The `reusable-domain-config.xml` file in the `WEB-INF/config` directory may be edited to customize both the model and how it will appear in application views. For example, change the essential XML element for a few properties from true to false or from false to true, and then rerun the web application. In the Note concept, locate the only reference property and add the following XML element:

```
<referenceDropDownLookup>false</referenceDropDownLookup>
```

You may also want to edit the text that appears after the `=` sign in the `TravelApp.properties` file.

However, if you change the content of the XML configuration, be aware that the next time you generate the configuration from ModelibraModeler you will lose those changes. The same problem exists if you improve the text after the = sign in the TravelApp.properties file. In the next chapter, we will see how to handle this problem, which is an issue with all code generators.

Consult the info.html file in the WEB-INF/logs directory for various log messages. From time to time, you may want to delete the info.html file. It will be recreated for you with only new messages.

Finally, you will run the web application locally by selecting the Start class in the travel.wicket.app package and by running its main method. In a web browser, use the http://localhost:8081/ url, ignore the error message and click on the only link to see the home page of the default web application.

Summary

The WebLinks model is enlarged with the new Comment concept. There is still no relationship used in the model. The Eclipse project in this chapter has a different structure of directories, since it represents a web application. The web application is configured in the web.xml file in the WEB-INF standard directory. The embedded Jetty server, supported by its jar file in the WEB-INF/lib directory, is used to run the web application within the context of the Eclipse project. With only one application class the WebLinks model is made alive as a web application. The home page of the web application is simple with one link to the domain page that provides two links for the only model. The first link provides read only pages, while the second link allows for updates of entities. In both cases, the navigation starts with the entry concepts.

While the code for the DmEduc project is not generated, the code for the TravelImpression project is generated both at the model level and at the level of model views.

The next chapter will show how to select and order entities in Modelibra.

Questions

1. What is the purpose of the WEB-INF directory?
2. What kind of web server installation is required to run a web application from the context of the Eclipse project?
3. What specific Java classes are necessary to have a default web application based on a domain model?
4. What happens if you add a method in the generic Java class and then regenerate the code?

Exercises

Exercise 4.1.

For a new model with only one concept, create a new Eclipse project with all necessary files in order to be able to see the model as a default web application.

Exercise 4.2.

Repeat the previous exercise in a new Eclipse project but this time with the code generation.

Exercise 4.3.

Compare the two projects from the two previous exercises and make a short report about differences between the two versions.

Web Links

[CSS] Cascading Style Sheets

<http://www.w3schools.com/css/default.asp>

[Filter] Servlet Filters

<http://java.sun.com/products/servlet/Filters.html>

[JavaForge] Java Forge Repository

<http://www.javaforge.com/>

[Jetty] Jetty Web Server

<http://jetty.mortbay.org/jetty/>

[Servlet] Java Web Technology

<http://java.sun.com/products/servlet/>

[Wicket] Wicket Web Framework

<http://wicket.apache.org/>