SECTION III: Domain Model Relationships

Chapter 6: One-to-many Relationship

The objective of this chapter is to introduce a relationship between two concepts. There are several types of relationships [ERM]. The most common relationship type is one-to-many. In this chapter, the category property is transformed into the new Category concept and the one-to-many relationship between the Category concept and the Url concept is established. This means that a category may have many urls and that a url has one category. More precisely, a category may have from 0 to N urls and a url must have at least 1 and at most 1 category (exactly 1 category). A relationship has two directions between two relationship neighbors. Each neighbor direction has two relationship cardinalities (min..max). A direction with the max cardinality greater than one is called a child direction, and a direction with the max cardinality of one is called a parent direction.

When there are relationships between concepts in a domain model, not all concepts are entry points into the model. Concepts without parents are model entries. More precisely, entities of an entry concept are a starting point in model traversals both in the domain model and the web application. Entities of concepts with parents are reached through the parent entities. Exceptionally, a child concept may become an entry point with some additional configuration and code.

A concept may have, at most, one user oriented identifier that consists of properties and/or neighbors of the concept. A simple identifier has only one property. In an entry concept, all entities must have a unique value for the concept's identifier. However, in a non-entry child concept, the identifier is often unique only within the child parent. The Url concept's identifier consists of the name property and the category neighbor. Thus, a name of a web link must be unique only within its category.

When entities are displayed in a web application, they may be presented as a table of entities, as a list of entities, or as slides. In a list presentation, entities are displayed entity by entity, with a list of children for each child concept. In a slide presentation, a slide consists of a parent entity and child items. When entities are updated they are shown only as a table of entities.

Domain Model

The category property of the Url concept from the previous chapter is dropped and replaced by the new Category concept. There is also a one-to-many relationship between Category and Url. The Category concept is an entry point into the model and the Url concept is not. A url can be found only by finding first its category. The minimal cardinality of one from the Url concept to the Category concept assures that each url has its category. The Url.name property is unique only within its parent category (id in front of the name property and the category neighbor in ModelibraModeler).

The relationship between two concepts is created in a diagram of ModelibraModeler first by selecting the line icon, second by clicking on the title area of a parent concept followed by clicking on the title area of a child concept. If you make a mistake, you can always undo the action in ModelibraModeler. By default, the 0..N cardinalities are displayed close to the parent concept (a parent entity has from 0 to N child entities) and 1..1 cardinalities appear close to the child concept (a child entity hast exactly one parent entity). If concept names were properly entered by using the *Enter* key, the name of the parent-child direction (actually the child neighbor for Modelibra) is the name of the child concept in plural. Since the name indicates the name of the child neighbor, it starts with a small (lower) letter. Similarly, the name of the child-parent direction is the name of the parent concept that starts with a small letter. The names reflect the maximal cardinalities of the relationship directions. Note the position of minimal and maximal cardinalities in ModelibraModeler. Since a direction indicates a neighbor in Modelibra, and since a neighbor is a special property of the source concept, the cardinalities are positioned close to the source concept and not close to the destination concept.

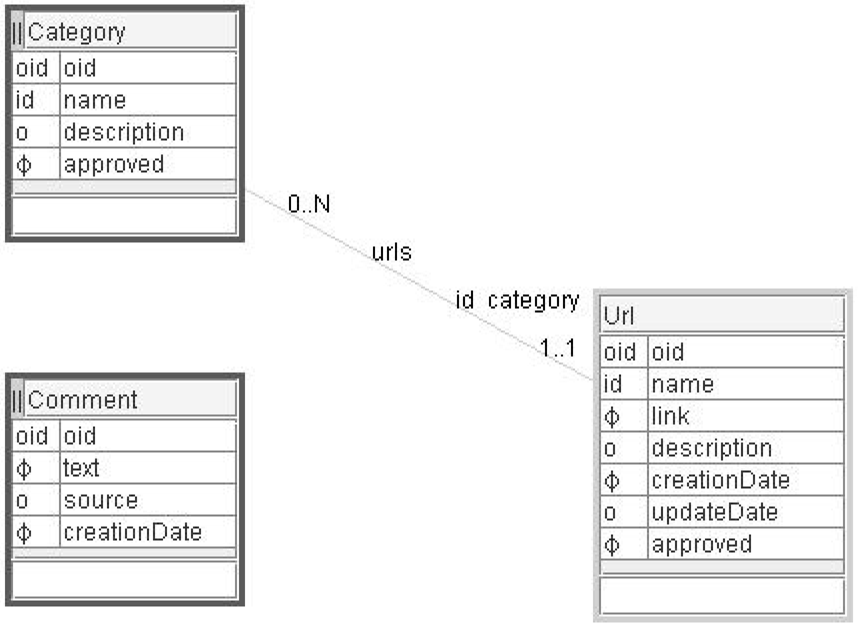


Figure 6.1. One-to-many relationship between Category and Url concepts

Domain Configuration

The specific-domain-config.xml file has been modified. The Url.category property has disappeared. The new Category concept definition is added and the Url concept definition has now a neighbor direction part.

```
<domains>

    <domain oid="1101">
          <code>DmEduc</code>
            <type>Specific</type>

          <models>

                 <model oid="110110">
                       <code>WebLink</code>
                       <author>Dzenan Ridjanovic</author>

                       <concepts>
```

```
<concept oid="110110100">
      <code>Category</code>
      <entitiesCode>Categories</entitiesCode>
      <entry>true</entry>
      <!-- model view -->
      <displayType>slide</displayType>

      <properties>
            <property oid="110110100110">
                  <code>name</code>
                  <propertyClass>
                        java.lang.String
                  </propertyClass>
                  <maxLength>64</maxLength>
                  <required>true</required>
                  <unique>true</unique>
            </property>
            <property oid="110110100120">
                  <code>description</code>
                  <propertyClass>
                        java.lang.String
                  </propertyClass>
                  <maxLength>510</maxLength>
            </property>
            <property oid="110110100130">
                  <code>approved</code>
                  <propertyClass>
                        java.lang.Boolean
                  </propertyClass>
                  <required>true</required>
                  <defaultValue>false</defaultValue>
            </property>
      </properties>

      <neighbors>
            <neighbor oid="110110100810">
                  <code>urls</code>
                  <destinationConcept>
                        Url
                  </destinationConcept>
                  <type>child</type>
                  <min>0</min>
                  <max>N</max>
            </neighbor>
      </neighbors>

</concept>

<concept oid="110110110">
      <code>Url</code>
      <entry>false</entry>
```

```
                        <properties>
                            ...
                            <property oid="110110110150">
                                <code>approved</code>
                                ...
                            </property>
                        </properties>

                        <neighbors>
                            <neighbor oid="110110110810">
                                <code>category</code>
                                <destinationConcept>
                                    Category
                                </destinationConcept>
                                <type>parent</type>
                                <min>1</min>
                                <max>1</max>
                                <unique>true</unique>
                            </neighbor>
                        </neighbors>

                    </concept>

                    <concept oid="110110120">
                        <code>Comment</code>
                        ...

                    </concept>

                </concepts>

            </model>

        </models>

    </domain>

</domains>
```

The Category concept is a domain model entry, and the Url concept is not. The Category display type is slide. This allows a display of category urls as in a slide presentation. Since the Category display type is slide, the Url display type is by default list, where Url properties are displayed as slide items in a list.

The Category concept has only one neighbor, the Url concept.

```
    <neighbors>
        <neighbor oid="110110100810">
            <code>urls</code>
            <destinationConcept>Url</destinationConcept>
            <internal>true</internal>
            <type>child</type>
            <min>0</min>
            <max>N</max>
        </neighbor>
    </neighbors>
```

The Category – Url direction has the urls code. The neighbor destination concept is Url. The direction is by default internal. This means that urls will be embedded internally within the parent category in the category.xml data file. The neighbor direction type is child. The direction min and max cardinalities are 0 and N respectively.

The Url - Category direction, with the category name (code in Modelibra), has the following definition:

```
    <neighbors>
        <neighbor oid="110110110810">
            <code>category</code>
            <destinationConcept>Category</destinationConcept>
            <internal>true</internal>
            <type>parent</type>
            <min>1</min>
            <max>1</max>
            <unique>true</unique>
        </neighbor>
    </neighbors>
```

The direction's neighbor concept is Category. The direction is internal. The direction type is parent with both min and max cardinalities of 1.

Concept Classes

The Category concept has, as usual, two specific classes, Categories and Category.

```
package dmeduc.weblink.category;

import java.util.Comparator;

import org.modelibra.Entities;
import org.modelibra.IDomainModel;
import org.modelibra.ISelector;
import org.modelibra.Oid;
import org.modelibra.PropertySelector;

public class Categories extends Entities<Category> {

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

    public Category getCategory(Oid oid) {
        return retrieveByOid(oid);
```

```
}

public Category getCategory(Long oidUniqueNumber) {
      return getCategory(new Oid(oidUniqueNumber));
}

public Category getCategory(String propertyCode, Object property) {
      return retrieveByProperty(propertyCode, property);
}

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

public Categories getCategories(String propertyCode, Object property) {
      return (Categories) selectByProperty(propertyCode, property);
}

public Categories getCategories(String propertyCode, boolean ascending) {
      return (Categories) orderByProperty(propertyCode, ascending);
}

public Categories getCategories(ISelector selector) {
      return (Categories) selectBySelector(selector);
}

public Categories getCategories(Comparator comparator, boolean ascending) {
      return (Categories) orderByComparator(comparator, ascending);
}

public Categories getApprovedCategories() {
      PropertySelector propertySelector = new PropertySelector("approved");
      propertySelector.defineEqual(Boolean.TRUE);
      return getCategories(propertySelector);
}

public Categories getNotApprovedCategories() {
      PropertySelector propertySelector = new PropertySelector("approved");
      propertySelector.defineEqual(Boolean.FALSE);
      return getCategories(propertySelector);
}

public Categories getKeywordCategories(String keyword) {
      PropertySelector propertySelector = new PropertySelector("description");
      propertySelector.defineContain(keyword);
      return getCategories(propertySelector);
}

public Categories getSomeKeywordCategories(String[] keywords) {
      PropertySelector propertySelector = new PropertySelector("description");
      propertySelector.defineContainSome(keywords);
      return getCategories(propertySelector);
```

```
	}

	public Categories getAllKeywordCategories(String[] keywords) {
		PropertySelector propertySelector = new PropertySelector("description");
		propertySelector.defineContainAll(keywords);
		return getCategories(propertySelector);
	}

	public Categories getCategoriesOrderedByName() {
		return getCategories("name", true);
	}

	public Categories getCategoriesOrderedByNameIgnoringCase() {
		CaseInsensitiveStringComparator caseInsensitiveStringComparator = new
			CaseInsensitiveStringComparator();
		PropertyComparator<Category> propertyComparator = new
			PropertyComparator<Category>("name",
				caseInsensitiveStringComparator);
		return getCategories(propertyComparator, true);
	}

	public Category createCategory(String name) {
		Category category = new Category(getModel());
		category.setName(name);
		if (!add(category)) {
			category = null;
		}
		return category;
	}

	public Category createCategory(String name, String description,
			boolean approved) {
		Category category = new Category(getModel());
		category.setName(name);
		category.setDescription(description);
		category.setApproved(approved);
		if (!add(category)) {
			category = null;
		}
		return category;
	}

}
```

Categories are sorted by the getCategoriesOrderedByName method. The CaseInsensitiveStringComparator class is used in the getCategoriesOrderedByNameIgnoringCase method to ignore whether there are small or capital letters in category names.

The Category class has a Java property for the urls neighbor (direction). Since the property represents the internal neighbor, it is initialized to empty entities in the Category constructor, linking the parent with the child neighbor. The property has usual set and get methods. The link between the child neighbor and the parent is established in the setUrls method.

```
package dmeduc.weblink.category;

import org.modelibra.Entity;
import org.modelibra.IDomainModel;

import dmeduc.weblink.url.Urls;

public class Category extends Entity<Category> {

	private String name;

	private String description;

	private Boolean approved = Boolean.FALSE;

	// Urls child neighbor (internal)
	private Urls urls;

	public Category(IDomainModel domainModel) {
		super(domainModel);
		// internal child neighbors only
		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 void setApproved(boolean approved) {
		setApproved(new Boolean(approved));
	}

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

	/*
	 * ** Neighbors ***
	 */

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

	public Urls getUrls() {
		return urls;
	}

}
```

The Urls class has a property for the internal parent neighbor. The parent property is initialized in the constructor with the Category argument. The property has the usual set and get methods.

```
	// internal parent neighbor
	private Category category;

	public Urls(IDomainModel model) {
		super(model);
	}

	public Urls(Category category) {
		this(category.getModel());
		// parent
		this.category = category;
	}

	/*
	 * Neighbors
	 */

	public void setCategory(Category category) {
		this.category = category;
	}

	public Category getCategory() {
		return category;
	}
```

The getCategoryUrls, getUrlsOrderedByCategory, getUrlsOrderedByCategoryIgnoringCategoryCase and getUrlsOrderedByCategoryThenName methods from the previous spiral have been removed in this spiral.

Similarly, the Url class has the internal parent neighbor property. The parent property is initialized in the constructor with the Category argument. The property has its set and get methods.

```
	// internal parent neighbor
    private Category category;

    public Url(IDomainModel model) {
        super(model);
        // internal child neighbors only
    }

    public Url(Category category) {
        this(category.getModel());
        // parent
        this.category = category;
    }

    /*
     * Neighbors
     */

    public void setCategory(Category category) {
        this.category = category;
    }

    public Category getCategory() {
        return category;
    }
```

Concept Tests

The tests in the UrlsTest class are adapted for the new Categories entry point into the model. The framework category is created and its urls are used in modified tests.

```
public class UrlsTest {

	public static final String FRAMEWORK_CATEGORY = "Framework";

	private static Categories categories;
	private static Category frameworkCategory;

	private static Urls frameworkCategoryUrls;

	@BeforeClass
	public static void beforeTests() throws Exception {
		categories = DmEducTest.getSingleton().getDmEduc().getWebLink()
				.getCategories();
```

```
        frameworkCategory = categories.createCategory(FRAMEWORK_CATEGORY);
        frameworkCategoryUrls = frameworkCategory.getUrls();
}

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

...

@Test
public void allUrls() throws Exception {
        frameworkCategoryUrls.createUrl(frameworkCategory, "Struts",
                        "http://struts.apache.org/", null, new EasyDate(2007, 4, 3),
                        false);
        frameworkCategoryUrls.createUrl(frameworkCategory, "Wicket",
                        "http://wicket.apache.org/", "Web component framework.",
                        new EasyDate(2008, 6, 22), true);
        frameworkCategoryUrls.createUrl(frameworkCategory, "Modelibra",
                        "http://www.modelibra.org/", "Domain model framework.",
                        new EasyDate(2008, 3, 27), true);
        Category personalCategory = categories.createCategory("Personal");

        assertNotNull(personalCategory);
        assertTrue(categories.contain(personalCategory));
        assertTrue(categories.getErrors().isEmpty());

        Urls personalCategoryUrls = personalCategory.getUrls();
        Url personalCategoryUrl = personalCategoryUrls.createUrl(
                        personalCategory, "Dzenan Ridjanovic",
                        "http://drdb.fsa.ulaval/");

        assertNotNull(personalCategoryUrl);
        assertTrue(personalCategoryUrls.contain(personalCategoryUrl));
        assertTrue(personalCategoryUrls.getErrors().isEmpty());

        WebLink webLink = DmEducTest.getSingleton().getDmEduc().getWebLink();
        Urls allUrls = webLink.getUrls();

        assertNotNull(allUrls);
        assertEquals(allUrls.size(), frameworkCategoryUrls.size()
                        + personalCategoryUrls.size());

        categories.remove(personalCategory);

        assertFalse(categories.contain(personalCategory));
}

@After
public void afterTest() throws Exception {
        for (Url url : frameworkCategoryUrls.getList()) {
```

```
				frameworkCategoryUrls.remove(url);
			}
		}

		@AfterClass
		public static void afterTests() throws Exception {
			categories.remove(frameworkCategory);

			DmEducTest.getSingleton().close();
		}

}
```

In the WebLink model class, there is a specific method to select all urls from the model. This method is tested in the allUrls test. Note the removal of the personalCategory object at the end of the test. Put it in the comment and try this test again.

Review all tests in the CategoriesTest class. You should not have problems in understanding them all. The following is a short summary of how to run JUnit tests in Eclipse.

Tests are executed in Eclipse by selecting the test class and the *Run As/JUnit Test* pop-up menu item. The result of the tests can be seen in the JUnit *tab*. If all tests are in *green*, they are successful. If there is the *blue* color, a test failure happened. The failure must be examined to understand why the expected result was not produced. If there is the *red* color, an error exception was raised.

All JUnit tests, found in the project, can be executed in Eclipse by selecting the project and the *Run As/JUnit Test* pop-up menu item.

Web Application

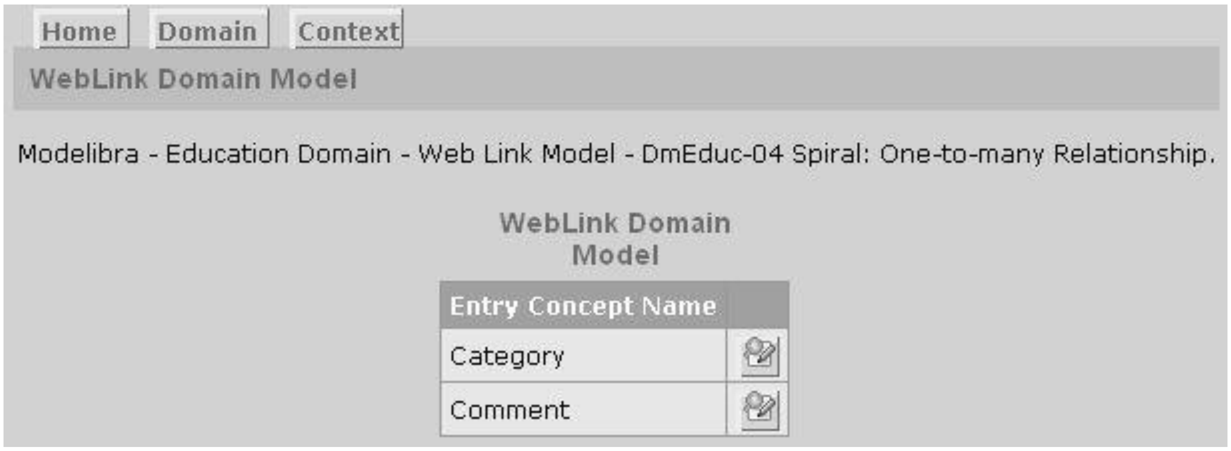
The DmEducApp.properties file has new keys for the Category concept.

```
Category=Category
Categories=Categories
Category.name=Category Name
Category.description=Description
Category.approved=Approved
Category.urls=Urls
Category.id=[Category Name]
Category.id.unique=[Category Name] must be unique.
Category.name.length=Category Name is too long.
Category.name.required=Category Name is required.
Category.description.length=Description is too long.
Category.approved.required=Approved is required.
```

There are also new keys for the Url concept for the Category neighbor and the new id.

```
Urls.category=Category
Url.id=[Category+Web Link Name]
Url.id.unique=[Category+Web Link Name] must be unique.
```

There are two entries into the model: the Category and Comment concepts (Figure 6.2).



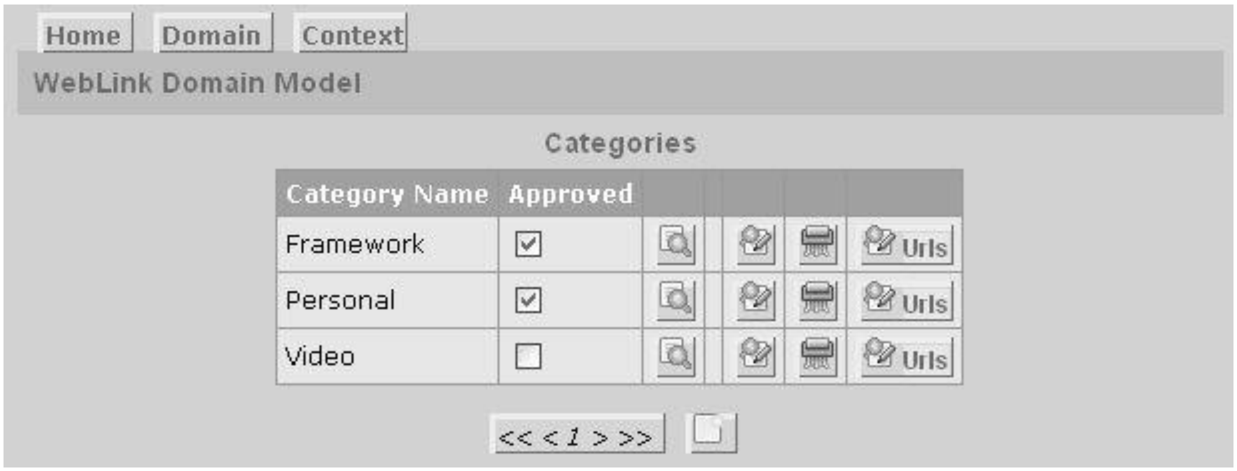
Modelibra - Education Domain - Web Link Model - DmEduc-04 Spiral: One-to-many Relationship.

WebLink Domain Model

Entry Concept Name
Category
Comment

Figure 6.2. Two entries

All categories, approved and not approved, are shown in the update page (Figure 6.3). Categories are ordered by name.



Category Name	Approved
Framework	✓
Personal	✓
Video	☐

Figure 6.3. All categories ordered by name

However, the not approved Video category is the first entity in the category.xml data file, because it was added first.

```
<categories>
  <category oid="1162133985304">
    <name>Video</name>
    <approved>false</approved>
    <urls>
      <url oid="1162134007634">
        <name>Google Video</name>
        <link>http://video.google.ca/</link>
        <creationDate>2007-01-03</creationDate>
        <approved>false</approved>
      </url>
    </urls>
  </category>
...
</categories>
```

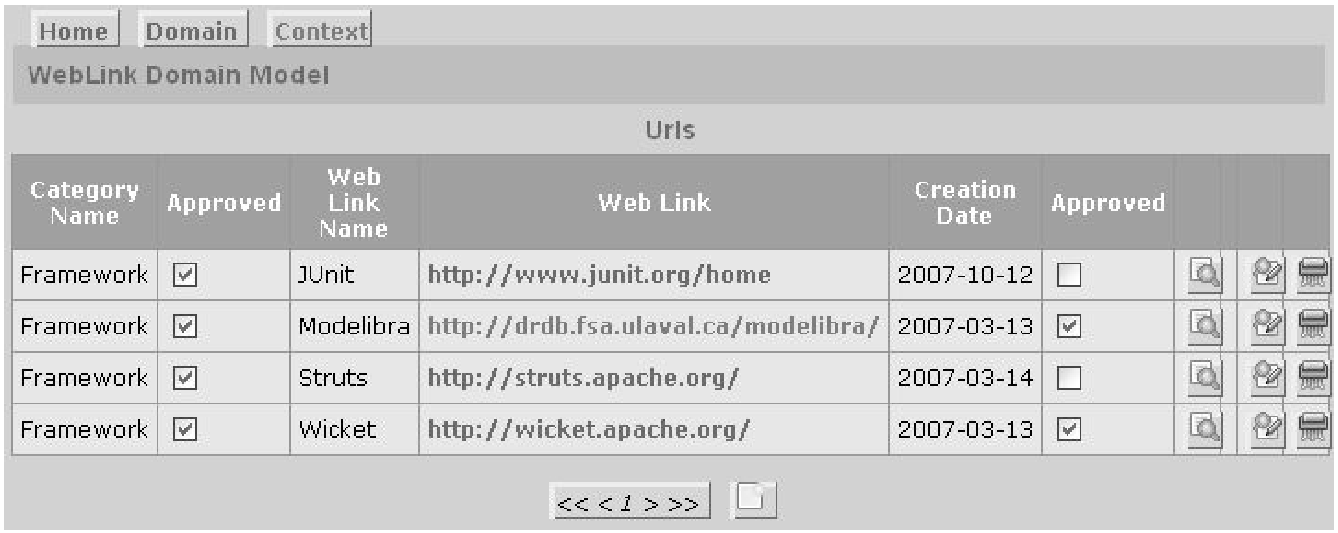
The order of categories is done in the specific view class called EntityUpdateTablePage from the dmeduc.wicket.weblink.category package. The class name is generic but it is located in the specific package. The class extends the class with the same name from the org.modelibra.wicket.concept generic package. Without the specific class, with the generic name, and without the following line

```
      categories = categories.getCategoriesOrderedByName();
```

in the getNewViewModel method of the EntityUpdateTablePage specific class, categories would have been shown in they same order as they appear in the category.xml data file. The generic class from the generic package is the web page component from ModelibraWicket. All web components in ModelibraWicket have the same two parameters. The first parameter is of the ViewModel type and the second parameter is of the View type. Thus, a web component in ModelibraWicket has its view model and its view. The view model is there to feed the view with necessary data that come from the model. The constructor of the specific EntityUpdateTablePage component passes the same view to the generic component. However, the view model is modified by the static getNewViewModel method. First, the new view model is constructed based on the given view model. The view model has current entities that can be reached by the getEntities method. Those entities are categories that are then ordered by the name property. The ordered categories are set as the current entities of the new view model and the new view model is returned to the constructor. Hence, the constructor of the generic web component accepts the new view model with ordered categories. When the generic EntityUpdateTablePage component is used in ModelibraWicket, the ModelibraWicket software first checks if there is a specific counterpart of that component.

```
package dmeduc.wicket.weblink.category;

import org.modelibra.wicket.view.View;
import org.modelibra.wicket.view.ViewModel;

import dmeduc.weblink.category.Categories;

public class EntityUpdateTablePage extends
            org.modelibra.wicket.concept.EntityUpdateTablePage {

      public EntityUpdateTablePage(final ViewModel viewModel, final View view) {
            super(getNewViewModel(viewModel), view);
      }

      private static ViewModel getNewViewModel(final ViewModel viewModel) {
            ViewModel newViewModel = new ViewModel(viewModel);
            Categories categories = (Categories) viewModel.getEntities();
            categories = categories.getCategoriesOrderedByName();
            newViewModel.setEntities(categories);
            return newViewModel;
      }
}
```

Since there is a one-to-many relationship between the Category and Url concepts, for a specific category in Figure 6.3 there is the *Urls* link displayed with the help of CSS as a button. If the Framework category is selected in Figure 6.3, by clicking on the *Urls* link, a new update page, with the urls of the Framework category, is displayed.



Category Name	Approved	Web Link Name	Web Link	Creation Date	Approved			
Framework	✓	JUnit	http://www.junit.org/home	2007-10-12	☐			
Framework	✓	Modelibra	http://drdb.fsa.ulaval.ca/modelibra/	2007-03-13	✓			
Framework	✓	Struts	http://struts.apache.org/	2007-03-14	☐			
Framework	✓	Wicket	http://wicket.apache.org/	2007-03-13	✓			

Figure 6.4. Urls of the "Framework" category

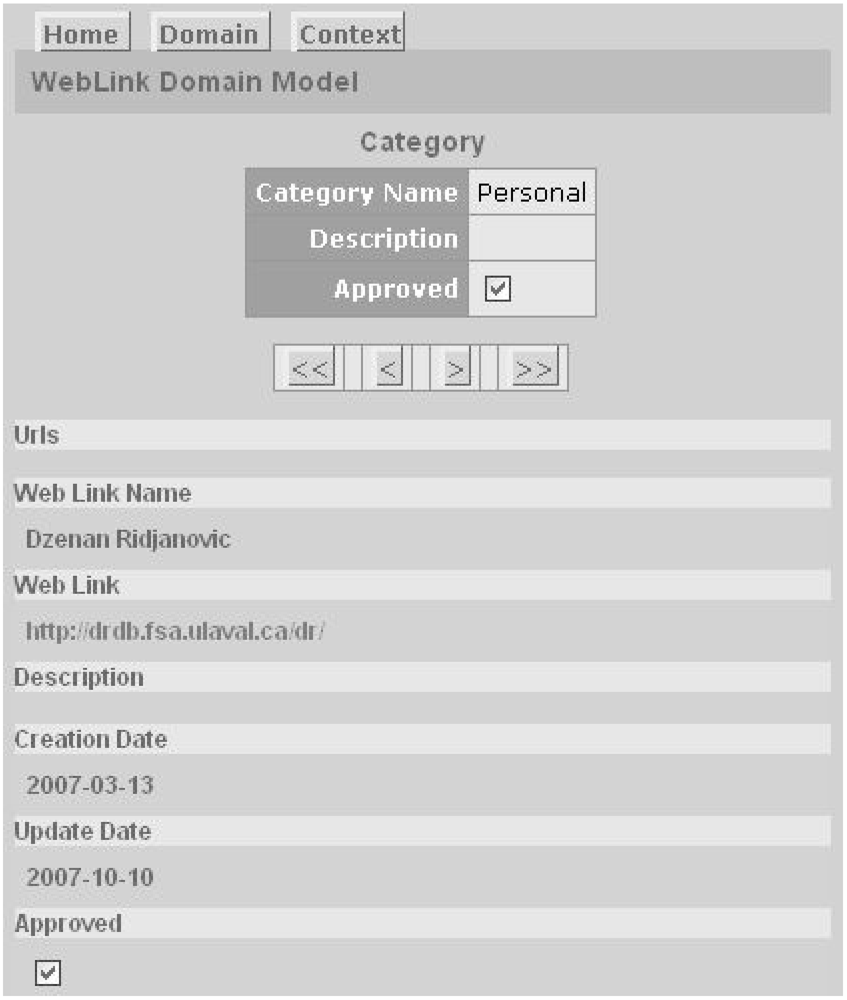
The Url entities, approved and not approved, are ordered by web link name. This is accomplished in the specific EntityUpdateTablePage component with the help of the getUrlsOrderedByName method. The component is specific because it is in the specific dmeduc.wicket.weblink.url package.

```
package dmeduc.wicket.weblink.url;

import org.modelibra.wicket.view.View;
import org.modelibra.wicket.view.ViewModel;

import dmeduc.weblink.url.Urls;

public class EntityUpdateTablePage extends
		org.modelibra.wicket.concept.EntityUpdateTablePage {

	public EntityUpdateTablePage(final ViewModel viewModel, final View view) {
		super(getNewViewModel(viewModel), view);
	}

	private static ViewModel getNewViewModel(final ViewModel viewModel) {
		ViewModel newViewModel = new ViewModel(viewModel);
		Urls urls = (Urls) viewModel.getEntities();
		urls = urls.getUrlsOrderedByName();
		newViewModel.setEntities(urls);
		return newViewModel;
	}
}
```

In the display mode, the categories are presented as slides. This is declared in the XML configuration file for the Category concept.

```
    <displayType>slide</displayType>
```

On the same display page, the current approved category with its approved urls is shown as a slide (Figure 6.5). The first current approved category is Framework. Since the Framework category has multiple urls, in order to save the chapter space, the next Personal category with only one url is shown here. Bellow the current category there is the slide bar with the links, displayed as buttons, for the first, prior, next and last entities.



WebLink Domain Model

Category

Category Name	Personal
Description	
Approved	✓

Urls

Web Link Name

Dzenan Ridjanovic

Web Link

http://drdb.fsa.ulaval.ca/dr/

Description

Creation Date

2007-03-13

Update Date

2007-10-10

Approved

✓

Figure 6.5. Personal category slide

Here, the entities displayed as slides are approved categories ordered by name. The selection and order of categories is done in the specific EntityDisplaySlidePage component by using the chain of two methods:

```
    categories = categories.getApprovedCategories().getCategoriesOrderedByName();
```

```
package dmeduc.wicket.weblink.category;

import org.modelibra.wicket.view.View;
import org.modelibra.wicket.view.ViewModel;

import dmeduc.weblink.category.Categories;

public class EntityDisplaySlidePage extends
		org.modelibra.wicket.concept.EntityDisplaySlidePage {


	public EntityDisplaySlidePage(final ViewModel viewModel, final View view) {
		super(getNewViewModel(viewModel), view);
	}

	private static ViewModel getNewViewModel(final ViewModel viewModel) {
		ViewModel newViewModel = new ViewModel(viewModel);
		Categories categories = (Categories) viewModel.getEntities();
		categories = categories.getApprovedCategories()
			.getCategoriesOrderedByName();
		newViewModel.setEntities(categories);
		return newViewModel;
	}

}
```

The category urls, approved and ordered by name, appear in the list section of the display category (as a slide) page. The display slide page is composed of two sections: the parent section for the category slide and the child section for the list of category urls. In Wicket, a page section is supported by the Panel component. Therefore, we cannot use the specific page component for the selection and order of urls. We need to use the specific EntityDisplayListPanel panel component in order to select only approved urls and then order them by name.

```
package dmeduc.wicket.weblink.url;

import org.modelibra.wicket.view.View;
import org.modelibra.wicket.view.ViewModel;

import dmeduc.weblink.url.Urls;

public class EntityDisplayListPanel extends
		org.modelibra.wicket.concept.EntityDisplayListPanel {

	public EntityDisplayListPanel(final ViewModel viewModel, final View view) {
		super(getNewViewModel(viewModel), view);
	}

	private static ViewModel getNewViewModel(final ViewModel viewModel) {
		ViewModel newViewModel = new ViewModel(viewModel);
		Urls urls = (Urls) viewModel.getEntities();
		urls = urls.getApprovedUrls().getUrlsOrderedByName();
		newViewModel.setEntities(urls);
		return newViewModel;
	}

}
```

In updates, only the table display type is used. In displays, either the table, list or slide display type can be chosen. The display type can be declared as the XML element of the concept. If nothing is declared, the default display type is table.

In the dmeduc.wicket.weblink.category package there are four specific web components with generic names. There are three specific components for the display mode and only one for the update mode. All possible specific web page components for the Category concept are there to allow you to experiment with the display of categories.

Modelibra Interfaces

There are no new methods used from the Modelibra interfaces.

Travel Impression

As a consequence of using the TravelImpression-02 spiral from the previous chapter, the model has changed in the new TravelImpression-03 spiral . There is a new concept called Url that is related to the Place parent concept in a one-to-many relationship. The Url concept has the name, webLink and description properties. The place neighbor and the name property form the identifier (id) of the Url concept. This means that two urls may have the same name but only if they belong to different places. In other words, all urls of the same place have the unique name. This restriction is more reasonable for users than the constraint that all urls of all places must have the unique name. The webLink property is mandatory but is not part of the identifier. Hence, the same web link may be used multiple times.

The Url concept has been introduced to provide more interesting web links for the same place. The Place concept has already the optional webLink property. If this web link is used it may be considered as the principal or the most important link for the place. In addition to this web link, other web links for the same place may be now entered. This may provide more useful information for the place to the users of the web application.

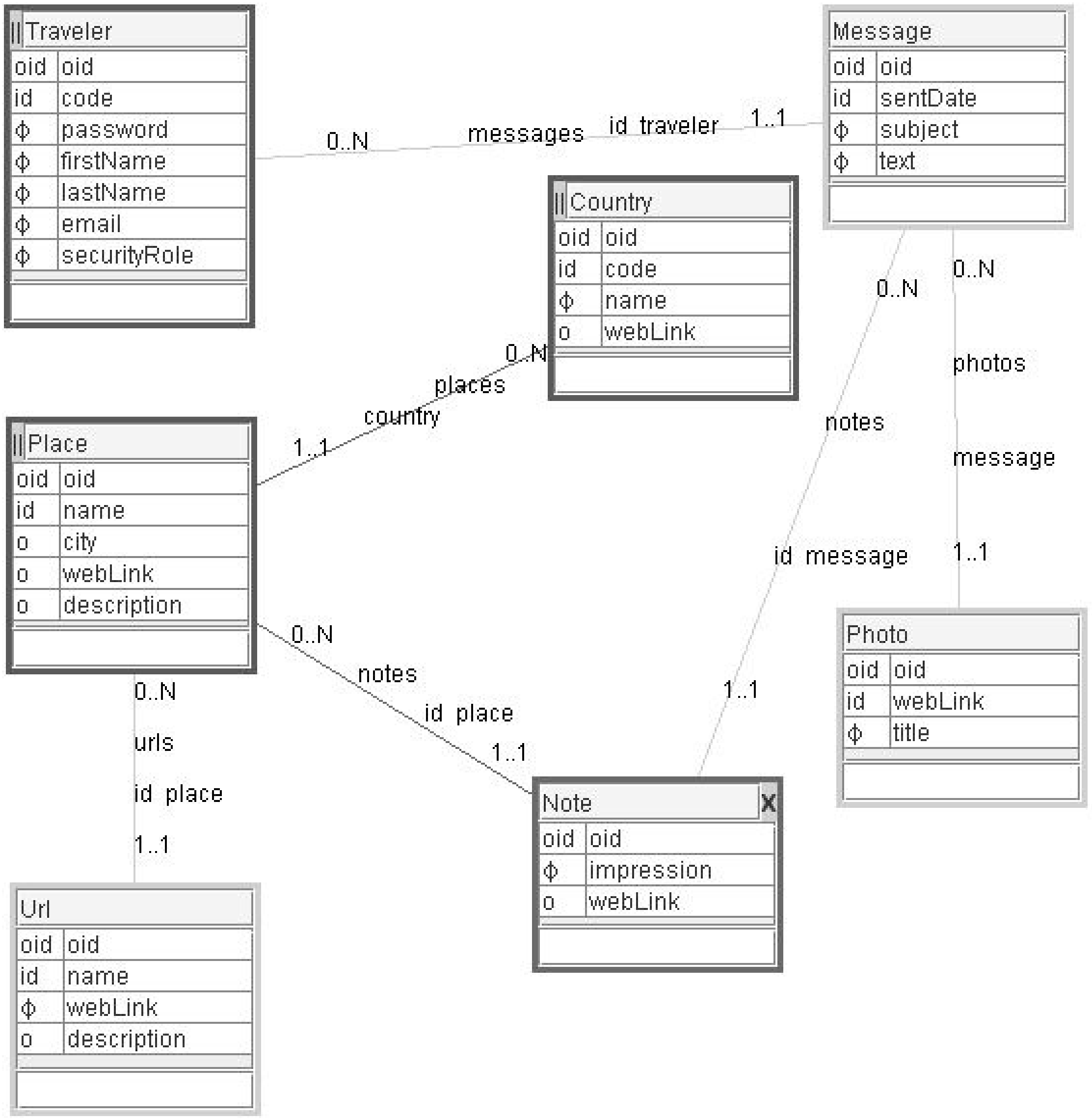


Figure 6.6. Travel Impression model

The new reusable XML configuration was generated from ModelibraModeler. This generation did not touch the specific XML configuration that had been introduced in the previous spiral. Based on the new reusable XML configuration and the specific XML configuration, the new code was generated partially, in order to keep specific changes unaltered.

The main method of the DmGenerator class in the dm.gen package, generates the code partially.

```
public static void main(String[] args) {
      DmGenerator dmGenerator = new DmGenerator();

      // dmGenerator.getDmModelibraGenerator().generate();
      // dmGenerator.getDmModelibraGenerator()
      //    .generateModelibraGenClasses();
      dmGenerator.getDmModelibraGenerator().generateModelibraPartially();

      // dmGenerator.getDmModelibraWicketGenerator().generate();
```

```
        // dmGenerator.getDmModelibraWicketGenerator()
        //    .generateModelibraWicketAppProperties();
        dmGenerator.getDmModelibraWicketGenerator()
            .generateModelibraWicketPartially();
    }
```

The Modelibra code was generated by the generateModelibraPartially method of the DmModelibraGenerator class in the dm.gen package. Since the JUnit tests have been introduced in this chapter and since the code generation of all test classes have been changed, all test classes were generated. For the new Url concept all five possible concept classes were generated. For the Place concept, the GenPlace class was regenerated to reflect the new property of the Urls type. Note that the if statement replaced the commented generation of the five classes for a concept.

```
    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, codeDirectoryPath);
            // modelGenerator.generateGenModel();
            // modelGenerator.generateModel();
            modelGenerator.generateModelTest();

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

                if (conceptConfig.getCode().equals("Place")) {
                    modelGenerator.generateGenEntity(conceptConfig);
                    modelGenerator.generateEntitiesTest(conceptConfig);
                }
                else if (conceptConfig.getCode().equals("Url")) {
                    modelGenerator.generateGenEntity(conceptConfig);
                    modelGenerator.generateGenEntities(conceptConfig);
                    modelGenerator.generateEntity(conceptConfig);
                    modelGenerator.generateEntities(conceptConfig);
                    modelGenerator.generateEntitiesTest(conceptConfig);
                } else {
                    modelGenerator.generateEntitiesTest(conceptConfig);
                }
```

```
			}

			// modelGenerator.generateEmptyXmlDataFiles();
		}
	}
```

The ModelibraWicket code was generated by the generateModelibraWicketPartially method of the DmModelibraWicketGenerator class in the dm.gen package. The domain application properties were regenerated, this time with the addition of properties for the new Url concept and the new Place property: Place.urls=Urls. In addition, four web components were generated for the new Url concept. These four specific web components with generic names will be used in the next spiral to provide selection and order for display and update of entities Note that the if statement replaced the commented generation of the four web components for a concept.

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

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

		for (ModelConfig modelConfig : domainConfig.getModelsConfig()) {

			for (ConceptConfig conceptConfig : modelConfig.getConceptsConfig()){
				// domainWicketGenerator
				//    .generateDomainModelConceptUpdateTablePage(conceptConfig);
				// domainWicketGenerator
				//    .generateDomainModelConceptDisplayTablePage(conceptConfig);
				// domainWicketGenerator
				//    .generateDomainModelConceptDisplayListPage(conceptConfig);
				// domainWicketGenerator
				//    .generateDomainModelConceptDisplaySlidePage(conceptConfig);

				if (conceptConfig.getCode().equals("Url")) {
					domainWicketGenerator
					.generateDomainModelConceptUpdateTablePage(conceptConfig);
					domainWicketGenerator
					.generateDomainModelConceptDisplayTablePage(conceptConfig);
					domainWicketGenerator
					.generateDomainModelConceptDisplayListPage(conceptConfig);
					domainWicketGenerator
					.generateDomainModelConceptDisplaySlidePage(conceptConfig);
				}
			}

		}
	}
```

Summary

The new Category concept is introduced to allow the use of the one-to-many relationship between the Category and Url concepts. As a consequence, the XML configuration is modified to include the XML declaration for the Category concept and the neighbor XML element for the Url concept. The Category and Categories classes are created and the Url and Urls classes are modified to include the new neighbor. Some tests are modified and some are added to explore the traversal of the one-to-many relationship.

The DmEduc web application has the new child link in the parent entities. The selection and order of entities is done in the specific classes that have the generic name to allow ModelibraWicket to intercept specific changes, without creating specific web components from scratch.

The TravelImpression model is changed to show the partial code generation that introduces the changes without destroying the specific changes done in the previous spiral. This is done for XML configurations, Java classes and application properties.

The next chapter will explain how reflexive and optional relationships of the one-to-many type are handled both in Modelibra and ModelibraWicket.

Questions

1. Without undoing the creation of a one-to-many relationship in ModelibraModeler, is it possible to change the cardinalities and neighbor names? If so, explain how it can be done.

2. What are the consequences if you remove a part of the JUnit test under the @Before annotation?

3. How can JUnit support the team work on the same Eclipse project?

4. What happens in the web application if you remove the parent neighbor and keep the child neighbor of the same relationship?

5. What happens in the web application if all specific web components with generic names are deleted?

6. Why bother with the partial code generation?

Exercises

Exercise 6.1.

Make a JUnit test to create a new category with its urls.

Exercise 6.2.

Make a JUnit test to select all urls that are created after a certain date.

Exercise 6.3.

Write a pre action in the Urls class and test it.

Web Links

[ERM] Entity-Relationmship Model
http://en.wikipedia.org/wiki/Entity-relationship_model