Chapter 3: Persistent Domain Model

The objective of this chapter is to create a persistent domain model. The model changes are saved in an XML data file without a user intervention. In Modelibra, the model is by default persistent, the persistence type is xml and the model is loaded and saved automatically by the framework.

In addition, steps to create a new domain model are discussed. A model is designed for a new project about travel impressions. In several model spirals, the model progressed from the initial thought to a reasonable model version.

Domain Model

The domain model still has only one concept: Url, although there are more properties than in the previous spiral. An entity of the Url concept must be classified by the required category property. The link property becomes the identifier (id) for the concept. This means that two different entities of the same Url concept cannot have the same value for the link property. The difference between the oid and the id of the concept is in the nature of identifier. The oid is an artificial identifier without any meaning for a user and is generated by ModelibraModeler. Usually, the oid values are hidden from users. The id is a semantical identifier with the real meaning for a user, and as such it is required (not null). A user identifier may be simple, consisting of one property, or composite, consisting of more than one property.

There are two new properties of the Date class (java.util.Date), one to indicate a date when a new url was created and another to show the last date when the url has been modified. The creationDate is required and its default value is determined by the date when a url is created. A url is either approved for public display or not. It is required and its default value is false. The approved property is of the Boolean class. In Modelibra, a concept's property must have a data type that is a Java class and not a base (primitive) type such as boolean. However, for those properties that have a class that has the corresponding base type, additional convenience methods with base types may be defined. For example, the isApproved method is defined in addition to the getApproved method.

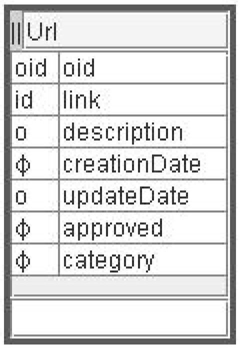
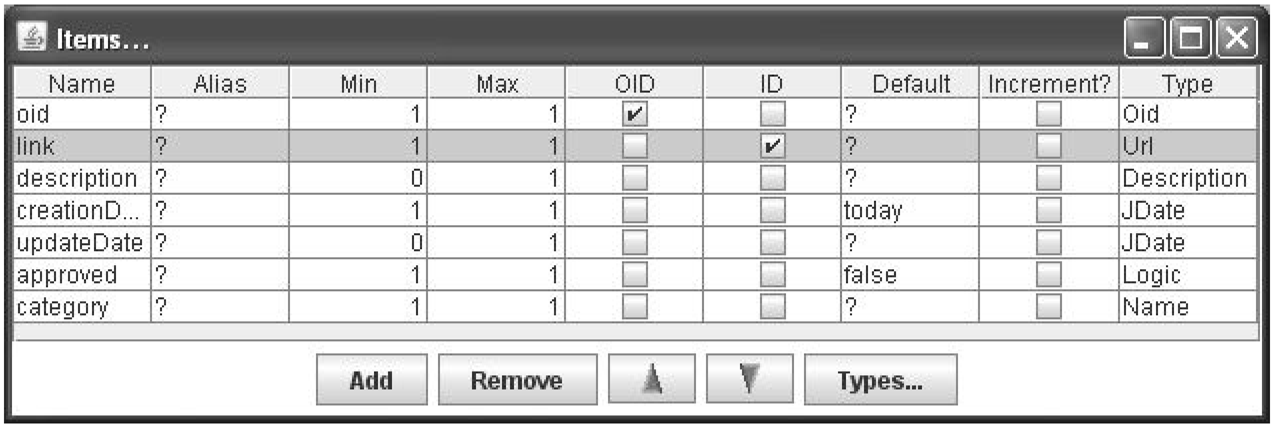


Figure 3.1. Url concept

Modelibra Modeler

Since the Url concept is still the entry into the model, the || sign, in front of the concept name, is used in ModelibraModeler. In addition to the artificial object identifier, the user oriented identifier, indicated by the id sign in front of the link property, is created by double-clicking on the sign space. The concept box in ModelibraModeler is divided in three areas: upper, middle and lower. The upper area is also called a title area. It has three sub-sections: the sign sub-section on the left, the title sub-section in the middle, and the sign sub-section on the right. The sign sub-section on the right is not visible in this spiral. The middle area has also two sub-sections: the sign sub-section and the properties sub-section. The lower area is used to add a new property.

After the new properties are entered by using the lower area, the pop-up menu is obtained by a right click on the title area. A window with a table of properties is obtained by the *Items...* selection (Figure 3.2). For each required property its minimal cardinality is changed from 0 to 1 in the *Min* column of the table. If there is a default value for a property it is entered in the *Default* column. The sequence of properties may be changed by the *upper* and the *lower arrow* buttons. The selected property may be removed by the *Remove* button. A new property may be created by the *Add* button. If a property is added in this window, the corresponding box in the diagram will show the new property. Thus, results of user actions are synchronized with the graphical display.

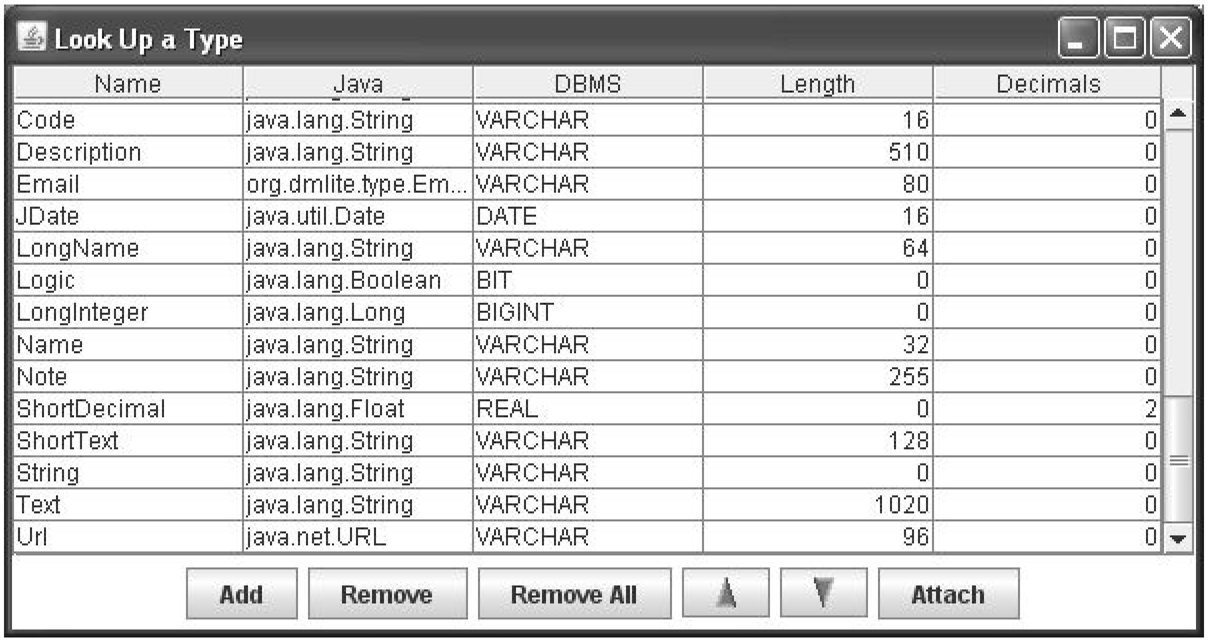


Name	Alias	Min	Max	OID	ID	Default	Increment?	Type
oid	?	1	1	✓		?		Oid
link	?	1	1		✓	?		Url
description	?	0	1			?		Description
creationD...	?	1	1			today		JDate
updateDate	?	0	1			?		JDate
approved	?	1	1			false		Logic
category	?	1	1			?		Name

Figure 3.2. Properties window

The selected property is typed by clicking on the *Types...* button. A type is looked up in the new window in Figure 3.3 by selecting the appropriate type name and clicking on the *Attach* button. The property will have a new type in the *Type* column in Figure 3.2.

In the table of types there are more rows than shown in Figure 3.3. Those types that are shown are user oriented types. Each type is defined by its semantical name, by the full Java class, by the corresponding JDBC type of a relational database management systems (DBMS), by its length and the number of decimals if appropriate. For example, the Description user type corresponds to the java.lang.String class, has the VARCHAR database type, and its maximal length is 510 characters. In comparison, the ShortDecimal type is of the java.lang.Float class, has the REAL database type, its length is not restricted as indicated by 0, and it has 2 decimal paces. A new user type may be added by the *Add* button and by entering required values.



Name	Java	DBMS	Length	Decimals
Code	java.lang.String	VARCHAR	16	0
Description	java.lang.String	VARCHAR	510	0
Email	org.dmlite.type.Em...	VARCHAR	80	0
JDate	java.util.Date	DATE	16	0
LongName	java.lang.String	VARCHAR	64	0
Logic	java.lang.Boolean	BIT	0	0
LongInteger	java.lang.Long	BIGINT	0	0
Name	java.lang.String	VARCHAR	32	0
Note	java.lang.String	VARCHAR	255	0
ShortDecimal	java.lang.Float	REAL	0	2
ShortText	java.lang.String	VARCHAR	128	0
String	java.lang.String	VARCHAR	0	0
Text	java.lang.String	VARCHAR	1020	0
Url	java.net.URL	VARCHAR	96	0

Figure 3.3. Types window

The types may also be reached by the *Dictionary* menu of the main window of ModelibraModeler. It is essential to export types and the diagram by using the *Transfer* menu, in order to have the text format for the domain model. If for any reason a model file cannot be opened, types and diagrams in the text format may be used to recuperated the model.

Domain Configuration

The specific-domain-config.xml file is modified. The persistence element with the false value is deleted, since the default value for the persistence of the model is true. The author element indicates the the author of the WebLink model is Dzenan Ridjanovic. Declarations of the new properties are added.

```
<?xml version="1.0" encoding="UTF-8"?>

<domains>

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

           <models>

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

                     <concepts>
```

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

      <properties>
            <property oid="110110110110">
                  <code>link</code>
                  <propertyClass>
                        java.lang.String
                  </propertyClass>
                  <validateType>true</validateType>
                  <validationType>
                        java.net.URL
                  </validationType>
                  <maxLength>96</maxLength>
                  <required>true</required>
                  <unique>true</unique>
            </property>
            <property oid="110110110120">
                  <code>description</code>
                  <propertyClass>
                        java.lang.String
                  </propertyClass>
                  <maxLength>510</maxLength>
            </property>
            <property oid="110110110130">
                  <code>creationDate</code>
                  <propertyClass>
                        java.util.Date
                  </propertyClass>
                  <maxLength>16</maxLength>
                  <required>true</required>
                  <defaultValue>today</defaultValue>
            </property>
            <property oid="110110110140">
                  <code>updateDate</code>
                  <propertyClass>
                        java.util.Date
                  </propertyClass>
                  <maxLength>16</maxLength>
            </property>
            <property oid="110110110150">
                  <code>approved</code>
                  <propertyClass>
                        java.lang.Boolean
                  </propertyClass>
                  <required>true</required>
                  <defaultValue>false</defaultValue>
            </property>
            <property oid="110110110160">
                  <code>category</code>
                  <propertyClass>
```

```
                                                      java.lang.String
                                                </propertyClass>
                                                <maxLength>32</maxLength>
                                                <required>true</required>
                                           </property>

                                      </properties>

                               </concept>

                           </concepts>

                    </model>

                </models>

           </domain>

      </domains>
```

The following elements with default values could have been added with the same result:

```
      <model oid="110110">
           ...
           <persistenceType>xml</persistenceType>
           <persistenceRelativePath>
                data/xml/dmeduc/weblink
           </persistenceRelativePath>
           <defaultLoadSave>true</defaultLoadSave>
           ...
           <concept oid="110110110">
                ...
                <fileName>url.xml</fileName>
```

This means that the persistence type is by default xml and that the data will be saved in the weblink directory of the data/xml/dmeduc/weblink path local to the project's root directory. The dmeduc directory corresponds to the DmEduc domain and the weblink directory corresponds to the WebLink model. Of course, another data path may be defined. The entities will be converted to the XML data and saved automatically by Modelibra in the url.xml file. Therefore, there will be no explicit use of the save method in the code. The same is true for loading the data when the application is started. The data will be loaded from the url.xml file and converted automatically into the domain model by Modelibra.

If a data path is not in the context of the project, the explicit use of the load and save methods of an xml persistent class from the Modelibra project is required. For example, the XmlEntities class in the org.modelibra.persistency.xml package has the load and save methods.

The link property is required, unique and has a new validation type: java.net.URL. The length of the property is limited to 96 characters.

```
      <property oid="110110110110">
           <code>link</code>
```

```
        <propertyClass>java.lang.String</propertyClass>
        <validateType>true</validateType>
        <validationType>java.net.URL</validationType>
        <maxLength>96</maxLength>
        <required>true</required>
        <unique>true</unique>
    </property>
```

It may appear strange that the link property is of the String class and that it has an additional validation by declaring the URL class. For the xml data, the URL class could have been used directly as the property class and the validation type could have been avoided. However, since the URL class is not recognized by relational DBMS as a valid database type, the String class, which all DBMS can recognize in the form of the VARCHAR type, is used for the property class and the additional validation is done by Modelibra.

The description property is not required by the fact that the required element is omitted. Its maximal length is determined in the XML configuration to be 510 characters. By default it would have been 32. Both dates are of the java.util.Date class. Even when working with a relational database, Modelibra does not use the SQL Date type (java.sql.Date). The creationDate property is required and its default value is today, which means the date when the object is created. The approved property is of the java.lang.Boolean class. It is required and its default value is false. The category property is required and limited to the length of 32 characters.

There are four different categories of validations that Modelibra will do when an entity is added or updated: unique, required, validation type and maximal length. All four of them will be tested in this spiral.

Eclipse Project

The spiral in this chapter is called DmEduc-01 and it is initially created by copying the DmEduc-00 spiral.

First, a zip file is produced by using the *File/Export.../General/Archive File* action path in Eclipse. The DmEduc-00.zip name is given to the archive file. If the DmEduc-00 spiral is linked to the Subversion (SVN) repository, the zip file will not have the directories and files created by Subversion.

Second, the file is unzipped in a workspace of Eclipse. Both the root directory and the project name were changed to DmEduc-01 before importing the project into Eclipse: *File/Import.../GeneralExisting Projects into Workspace*.

Another way to export a project is to use the *File/Export.../General/File System* action path in Eclipse. The whole project would be copied to another location on your computer without the directories and files created by Subversion.

A new project is committed to the Subversion repository by the *Team/Share project.../SVN* pop-up menu item. The existing repository location is chosen and a folder name is browsed, usually under the trunk directory. A list of files to share is presented and not all of them are selected to be transferred to the repository. For example, the project classes take a lot of space and they can easily be recreated by

Eclipse on a local machine. Often, log files are also excluded from sharing.

If there is a difference between a local project version and a repository project, the local Eclipse project has a white star sign on the black background. In order to ignore directories and/or files from the local project version, the svn:ignore Subversion property is linked to the parent directory. For example, in the DmEduc-01 project, the classes directory is within the project's root directory. The SVN ignore property is defined by the *Team/Set Property...* pop-up menu item on the root directory:

```
	Property name:	svn:ignore
	Property content:	classes
				logs
```

Domain Classes

The PersistentDmEduc class extends the PersistenceDomain class from Modelibra. The only argument of the constructor is the domain object of the DmEduc class. The domain object does not have any knowledge of its persistence. This knowledge is separated into the new PersistentDmEduc class.

```
package dmeduc;

import org.modelibra.persistency.PersistentDomain;

public class PersistentDmEduc extends PersistentDomain {

	public PersistentDmEduc(DmEduc dmEduc) {
		super(dmEduc);
	}

}
```

The PersistentDmEduc class is used in the DmEducTest class in the test source directory. The persistent domain is a new property that is created in the open method of the domain test class. When the concept tests are done, the persistent domain is closed by calling the close method, in order to clean up persistent resources that are expensive to keep around for a long time.

```
package dmeduc;

import org.modelibra.config.DomainConfig;

public class DmEducTest {

	private static DmEducTest dmEducTest;

	private DmEduc dmEduc;

	private PersistentDmEduc persistentDmEduc;

	private DmEducTest() {
		super();
		open();
	}

	public static DmEducTest getSingleton() {
		if (dmEducTest == null) {
			dmEducTest = new DmEducTest();
		}
		return dmEducTest;
	}

	private void open() {
		DmEducConfig dmEducConfig = new DmEducConfig();
		DomainConfig domainConfig = dmEducConfig.getDomainConfig();
		dmEduc = new DmEduc(domainConfig);
		persistentDmEduc = new PersistentDmEduc(dmEduc);
	}

	public DmEduc getDmEduc() {
		return dmEduc;
	}

	public void close() {
		if (persistentDmEduc != null) {
			persistentDmEduc.close();
		}
	}

}
```

Concept Classes

The Url and Urls classes inherit the same Modelibra classes as in the previous spiral. There is nothing in those two classes that indicates that model objects may be saved. The new version of the Url class has the additional properties with the corresponding set and get methods.

```
package dmeduc.weblink.url;

import java.util.Date;

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

public class Url extends Entity<Url> {

	private String link;

	private String description;

	private Date creationDate;
private Date updateDate;

private Boolean approved;

private String category;

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

public void setLink(String link) {
      this.link = link;
}

public String getLink() {
      return link;
}

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

public String getDescription() {
      return description;
}

public void setCreationDate(Date date) {
      creationDate = date;
}

public Date getCreationDate() {
      return creationDate;
}

public void setUpdateDate(Date date) {
      updateDate = date;
}

public Date getUpdateDate() {
      return updateDate;
}

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();
	}

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

	public String getCategory() {
		return category;
	}

}
```

The link, description and category properties are of the String class. The link property is configured to have the URL validation type and the validation is done by Modelibra and not by Java. This is done to make the framework flexible with respect to different type requirements of DBMS. The creationDate and updateDate properties are of the Date class. The approved property is of the Boolean class. In Modelibra, a property must have a data type that is a Java class and not a base (primitive) type such as int or boolean. However, for those properties that have a class that has the corresponding base type, additional convenience methods with base types may be defined. For example, the approved property that is of the Boolean class, in addition to regular set and get methods,

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

	public Boolean getApproved() {
		return approved;
	}
```

has also the following convenience methods:

```
	public void setApproved(boolean approved) {
		setApproved(new Boolean(approved));
	}

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

The Urls class is more elaborate than the same class in the previous spiral. There are three new get methods.

```
package dmeduc.weblink.url;

import org.modelibra.Entities;
import org.modelibra.IDomainModel;
import org.modelibra.Oid;
```

```
public class Urls extends Entities<Url> {

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

	public Url getUrl(Oid oid) {
		return retrieveByOid(oid);
	}

	public Url getUrl(Long oidUniqueNumber) {
		return getUrl(new Oid(oidUniqueNumber));
	}

	public Url getUrlByLink(String link) {
		return retrieveByProperty("link", link);
	}

	public Url createUrl(String link, String category) {
		Url url = new Url(getModel());
		url.setLink(link);
		url.setCategory(category);
		if (!add(url)) {
			url = null;
		}
		return url;
	}

}
```

The three new retrieval methods find a url based on different arguments. All three methods are convenience methods since they delegate the real work to the methods inherited from the Entities class. The first two methods use an object identifier (oid) to find the corresponding entity. The getUrl method with the Oid argument is based on the retrieveByOid method. The Oid class is responsible for creating a new unique oid of the Long class, which is actually a time stamp with an additional counter to avoid time collisions. When an argument of the Oid class is not available but only the unique number, the getUrl method with the Long argument is used to find a url with that number. Note that there are two getUrl methods that differ only in the type of their arguments. This is an example of the method overloading [Overloading].

The third get method is based on a property. Its getUrlByLink name indicates that the property is link. The method that does the actual retrieval is retrieveByProperty. For the given property name and the property value, the first url that has that value for the property is returned. In all three retrieval methods, if an entity is not found null is returned.

There is also a new argument in the createUrl method to provide the required category name. The add method is inherited from the Entities class. It adds a new entity to a collection of entities, but only if all validations succeed. Thus, the result of the add method is either true or false. If at least one of the validations does not pass the test, the value is false and the url object is set to null.

JUnit Tests

The Url concept is tested in the UrlsTest class. In order to execute tests in Eclipse, the class is selected and in the pop-up menu the *Run As/JUnit Test* item is chosen. The JUnit tag leads us to the summary information about tests. All tests have passed.

Java annotations different from @Test indicate to JUnit what to do before and after tests. A concept has its own test class with JUnit tests. The tests are executed by selecting the test class in the *Java perspective* and by choosing the *Run As/JUnit Test* menu item in the pop-up menu.

Before all tests are executed, the entry point of the model is found. In this spiral there is only one concept in the model. Thus, there is only one entry point into the model. The concept is Url and a collection of urls represented by the Urls class will be used to test different actions.

```
private static Urls urls;

@BeforeClass
public static void beforeTests() throws Exception {
	urls = DmEducTest.getSingleton().getDmEduc().getWebLink().getUrls();
}
```

Before every test is executed a collection of errors for the entry point is emptied. Hence, if an error happens, we are sure that it is created by the current test.

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

After every test, the entry point is emptied by removing the created entities. Thus, each test starts from scratch and there are no dependencies introduced by the previous tests.

```
@After
public void afterTest() throws Exception {
	for (Url url : urls.getList()) {
		urls.remove(url);
	}
}
```

After all tests, the domain test object is closed in order to clean up the used resources.

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

Each test has a meaningful name that carries the essence of the test. Two urls with required values are created in the urlsRequiredCreated method. Different JUnit assertions are used to define expected outcomes of the creation of two urls. For pedagogical reasons, there are more assertions than necessary. For example, if the collection of entities contain the created entity, there is no need to assert that the entity is not null.

```
@Test
public void urlsRequiredCreated() throws Exception {
      Url url01 = urls.createUrl("http://www.modelibra.org/", "Framework");
      Url url02 = urls.createUrl("http://drdb.fsa.ulaval.ca/", "Personal");

      assertNotNull(url01);
      assertTrue(urls.contain(url01));
      assertNotNull(url02);
      assertTrue(urls.contain(url02));
      assertTrue(urls.getErrors().isEmpty());
}
```

For each required property there is one test that fails to create an entity without the required value. The error message can be found in the collection of errors for the collection of entities.

```
@Test
public void linkRequired() throws Exception {
      Url url = urls.createUrl(null, "Framework");

      assertNull(url);
      assertFalse(urls.contain(url));
      assertFalse(urls.getErrors().isEmpty());
      assertNotNull(urls.getErrors().getError("Url.link.required"));
}

@Test
public void categoryRequired() throws Exception {
      Url url = urls.createUrl("http://www.modelibra.org/", null);

      assertNull(url);
      assertFalse(urls.contain(url));
      assertFalse(urls.getErrors().isEmpty());
      assertNotNull(urls.getErrors().getError("Url.category.required"));
}
```

The uniqueness of an identifier for the Url concept is tested in the linkUnique method. An identifier may be composed of several properties. In that case the idUnique method name would be more appropriate.

```
@Test
public void linkUnique() throws Exception {
      String link = "http://drdb.fsa.ulaval.ca/";
      urls.createUrl(link, "Personal");
      Url notUniqueUrl = urls.createUrl(link, "Personal");

      assertNull(notUniqueUrl);
      assertFalse(urls.contain(notUniqueUrl));
```

```
		assertFalse(urls.getErrors().isEmpty());
		assertNotNull(urls.getErrors().getError("Url.id.unique"));
	}
```

The link property has the java.net.URL validation type. This validation is announced in the XML configuration file. When this property carries a String value that is not a url address, the validation error is Url.link.validation.

```
	@Test
	public void linkType() throws Exception {
		String link = "dzenan.ridjanovic@fsa.ulaval.ca";
		Url url = urls.createUrl(link, "Personal");

		assertNull(url);
		assertFalse(urls.contain(url));
		assertFalse(urls.getErrors().isEmpty());
		assertNotNull(urls.getErrors().getError("Url.link.validation"));
	}
```

The maximum length of a url value is declared in the XML configuration.

```
	@Test
	public void linkLength() throws Exception {
		String link =
"http://drdb.fsa.ulaval.ca/drdb.fsa.ulaval.ca/drdb.fsa.ulaval.ca/drdb.fsa.ulaval.ca/drdb.fsa.ulaval
.ca";
		Url url = urls.createUrl(link, "Personal");

		assertNull(url);
		assertFalse(urls.contain(url));
		assertFalse(urls.getErrors().isEmpty());
		assertNotNull(urls.getErrors().getError("Url.link.length"));
	}
```

The length test is also done for the category property, but this time within the context of an entity update.

```
	@Test
	public void categoryLengthUpdate() throws Exception {
		Url url = urls.createUrl("http://wicket.apache.org/", "Framework");
		Url urlCopy = url.copy();
		String category = "Component based Web frameworks -- components as building blocks.";
		urlCopy.setCategory(category);
		urls.update(url, urlCopy);

		assertFalse(urls.getErrors().isEmpty());
		assertNotNull(urls.getErrors().getError("Url.category.length"));
	}
```

When a url entity is created without a value of the creation date, a default value will be set by Modelibra as the today's date. This is also indicated in the XML configuration for the creationDate property.

```
@Test
public void creationDateDefault() throws Exception {
      Url url = urls.createUrl("http://drdb.fsa.ulaval.ca/", "Personal");
      Date creationDate = url.getCreationDate();

      assertNotNull(creationDate);

      Calendar calendar = Calendar.getInstance();
      calendar.setTime(new Date());
      int year = calendar.get(Calendar.YEAR);
      int month = calendar.get(Calendar.MONTH);
      int day = calendar.get(Calendar.DAY_OF_MONTH);
      int hour = calendar.get(Calendar.HOUR_OF_DAY);
      int minute = calendar.get(Calendar.MINUTE);
      calendar.setTime(creationDate);

      assertEquals(year, calendar.get(Calendar.YEAR));
      assertEquals(month, calendar.get(Calendar.MONTH));
      assertEquals(day, calendar.get(Calendar.DAY_OF_MONTH));
      assertEquals(hour, calendar.get(Calendar.HOUR_OF_DAY));
      assertEquals(minute, calendar.get(Calendar.MINUTE));
}
```

A default value for the approved property is declared in the XML configuration to be false.

```
@Test
public void approvedDefault() throws Exception {
      Url url = urls.createUrl("http://drdb.fsa.ulaval.ca/", "Personal");
      Boolean approvedObject = url.getApproved();
      boolean approved = url.isApproved();

      assertNotNull(approvedObject);
      assertFalse(approved);
}
```

A copy of the created entity is a different object that has the same content.

```
@Test
public void urlEquality() throws Exception {
      Url url = urls.createUrl("http://wicket.apache.org/", "Framework");
      Url urlCopy = url.copy();

      assertNotNull(urlCopy);
      assertEquals(url, urlCopy);
      assertNotSame(url, urlCopy);
}
```

The following method uses many different assertions for pedagogical reasons.

```
	@Test
	public void urlUpdate() throws Exception {
		Url url = urls.createUrl("http://wicket.apache.org/", "Framework");
		Url urlCopy = url.copy();
		urlCopy.setDescription("Component based Web framework.");
		urlCopy.setUpdateDate(new Date());

		assertNotNull(urlCopy);
		assertNotSame(url, urlCopy);
		assertEquals(url, urlCopy);
		assertTrue(url.equals(urlCopy));
		assertTrue(url.equalOid(urlCopy));
		assertFalse(url.equalProperties(urlCopy));

		urls.update(url, urlCopy);

		assertTrue(urls.getErrors().isEmpty());

		Url updatedUrl = urls.getUrl(urlCopy.getOid());

		assertNotNull(updatedUrl);
		assertNotSame(url, urlCopy);
		assertNotSame(updatedUrl, urlCopy);
		assertSame(url, updatedUrl);
		assertEquals(url, urlCopy);
		assertEquals(updatedUrl, urlCopy);
		assertEquals(url, updatedUrl);
	}
```

Modelibra Interfaces

In Modelibra, the Entity class implements the IEntity interface and the Entities class implements the IEntities interface. Since the Url class extends the Entity class and the Urls class extends the Entities class, the Url class inherits all methods declared in the IEntity interface, and the Urls class inherits all methods declared in the IEntities interface. In contrast with the class concept, the interface concept does not implement any methods.

Not all methods in the two interfaces are obvious. Many of them will be gradually introduced in the following spirals. The following methods have been used in the DmEduc-01 spiral for the first time: copy, update, retrieveByOid and retrieveByProperty.

```
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();

}
```

New Project

In this chapter, in parallel with the DmEduc-01 spiral, we will start another project to show how a new project starts (TravelImpression-00). This new project will allow readers to follow our approach in order to develop sooner their first project with Modelibra. With several subsequent chapters, the project will evolve, but in a way that reflects the usual project progression with Modelibra. This will allow more experienced readers to be more productive sooner. All new concepts and terms will be briefly introduced and we may return, with more detailed explanations, to some of them in future DmEduc spirals. Thus, the objective of the parallel project is not pedagogical as is the case with the DmEduc project. The real objective of the new project is to progress in spirals but in a way that developers of Modelibra do.

The domain of the new 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 loosing 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. Even, a traveler may send only an email message about an impression of visiting a certain place to one friend, perhaps with a few photos, and the friend may use the web application to present impressions about visited places, expressed in the message, to other interested people in a more informative and pleasing way. Of course, the traveler can do all of that without help from other people.

A domain model is also designed in spirals. The first spiral starts with the most important concepts from the chosen domain. In our Travel domain, the key concepts are: Traveler, Place and Impression. In general, a traveler visits many different places and may have an impression for each one. A place is something of interest for a traveler. It may be as general as a city, or as specific as a monument in a city. It may also be a village or a spot in nature. The important thing is that a traveler has an impression about the visited place to share with family and friends.

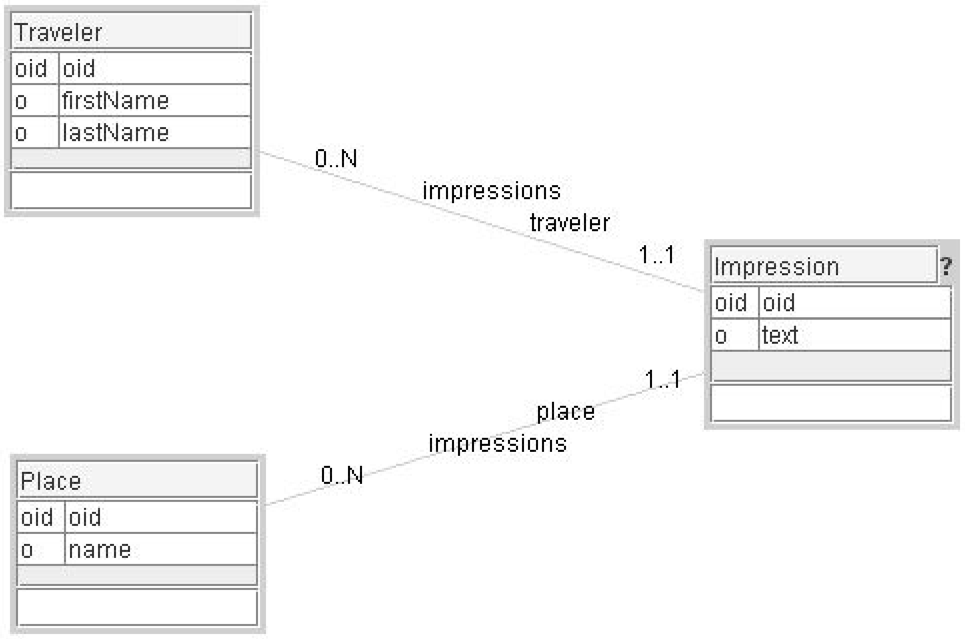


Figure 3.4. Travel Impression: Spiral 00

Figure 3.4. is the first cut model in ModelibraModeler. There are three concepts and two relationships. Each relationship has two directions. The direction from the Traveler concept to the Impression concept is also a neighbor for the Traveler concept. Thus, the Traveler concept has three properties and one neighbor. The properties are oid, firstName and lastName, and the neighbor is impressions. As in Java, the names of concepts, properties and neighbors are standardized. Since the concept corresponds to a class, its name starts with a capital letter. A property name starts with a small letter, and if a name is composed of several sub-names, each sub-name starts with a capital letter. Spaces are not used in names. Since a neighbor is a relationship property, its name starts with a small letter.

The meaning of the impressions neighbor for the Traveler concept is expressed in the following way: A traveler may have from 0 to N impressions. In the opposite direction, an impression has at least 1 and at most 1 traveler, or in a more natural way, an impression is given by exactly one traveler. Similarly, a place may be mentioned in many impressions and an impression is about exactly one place.

By default, each concept has its oid. By default, all properties may be null. The Impression concept has a question mark in the right section of the title area. This will be resolved in one of subsequent spirals of the model.

What happens if a traveler sends a rather long email message to his family, and if he describes more than one place in the same message. Would this be a common situation? We have to make a judgment here and the judgment is in favor of the situation. In Figure 3.5., there is a new spiral of the model with some improvements with respect to the previous spiral. In the spiral approach, each design or development iteration brings more clarifications and more details. If the reasoning behind the spirals is recorded in the documentation of the model, it would be easy for newcomers to the project to get familiar, in a step-by-step fashion, with the current version of the model.

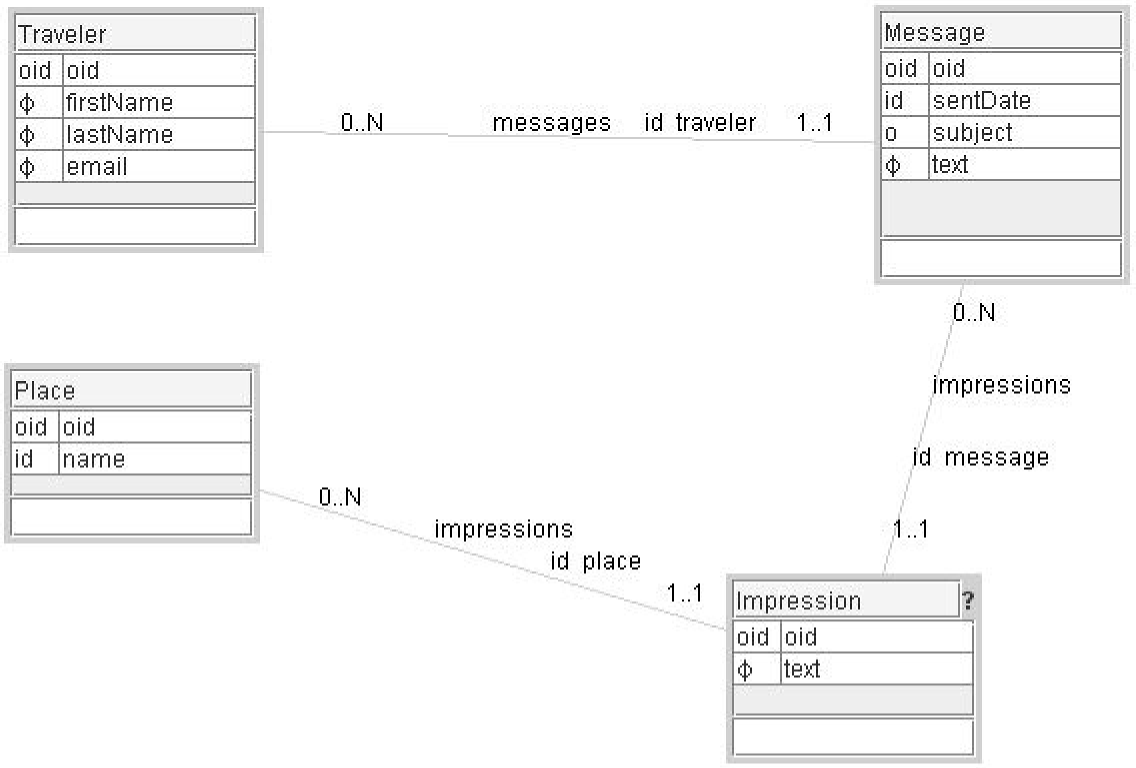


Figure 3.5 Travel Impression: Spiral 01

In ModelibraModeler, the new spiral is created in the application main window by selecting the previous spiral and making its copy by the *Copy* button. In this way, the history of spirals is preserved in the same mm file. The spiral model name will become Impression-01.

There is a new concept representing a message about impressions of visited places: Message. A traveler may send many messages on his trip. A message may contain impressions about several visited places. In probably the most common scenario, a traveler would either send an email, or would create a message directly on the web site, and someone from the circle of family and friends would enter specific impressions about visited places, by extracting portions of the message text.

A traveler has an email that would allow his family and friends to contact him for more personal communication. The judgment has been made that a traveler must have an email (as is the case for his names). A message has a date when it is sent (sentDate), a subject of the message is optional, and a full text of the message must be provided. An impression must have a text, and a place name is mandatory.

The user oriented identifier (id on a single property is a simple identifier) of the Place concept is its name. This means that each place name must be unique. An impression is identified by its source message and the place about which the impression is formed (ids on neighbors). Thus, the user oriented identifier of the Impression concept is composed of two neighbors. A message is identified by the traveler that sent the message and by the date when the message was sent (sentDate). For the time being, a traveler does not have a user oriented identifier. In Modelibra, the only identifier that is mandatory for a concept is oid. However, it is recommended to define ids, since they bring more meaning to the model. All properties are typed by Modelibra types (not seen in the graphical model). For example, a place name is of the LongName type and a traveler email is of the Email type.

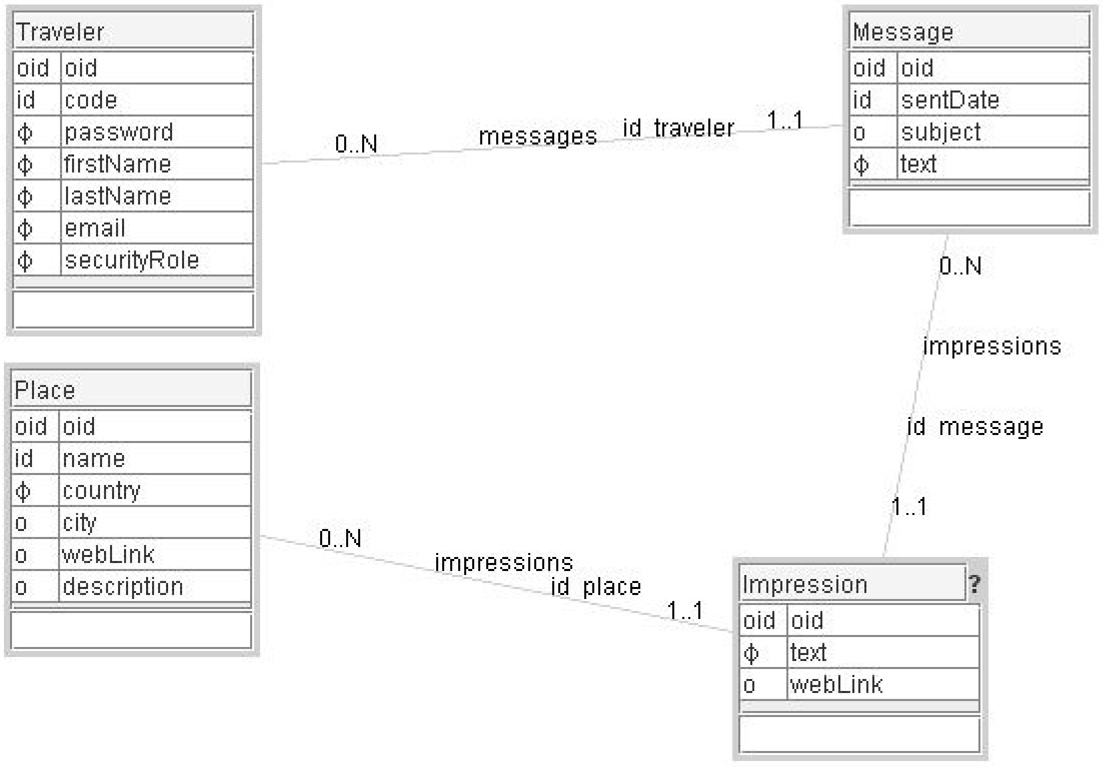


Figure 3.6 Travel Impression: Spiral 02

In Figure 3.6, the Traveler concept has an identifier that is the code property. A traveler selects a code that is of the String type with the maximal length of 16. The code must be provided and the traveler's password as well. A new traveler will have, by default, the “regular” value for the security role (securityRole). Different security roles will be used to provide different access and update privileges.

The Place concept is augmented with one mandatory property. A place must be located in a specific country. Often, a visited place will be in a city. However, there are many interesting places that are not located in cities. A short description may be entered about a new place. An additional web link (webLink) that is relevant for an impression about the place may be used.

In Modelibra there is one important restriction for a concept name. A concept name cannot be identical to either the domain name or the model name. Since the model is called Impression, we have to find a new name for the Impression concept. After a hesitation between PlaceImpression and Note, the Note name has been chosen in Figure 3.7. Since a property name can be the same as the model name, the new name for the text property in the Note concept is impression. Also, the former impressions name for two neighbors is changed to notes. This is accomplished by clicking on the *Dictionary* menu in the diagram window, then on the *Lines...* menu item. The new names can be found, together with some additions, in Figure 3.7.

A nice feature would be to upload photos made on the trip. The photos may be saved on a site such as Flickr [Flickr] and referenced from the Travel Impression application.

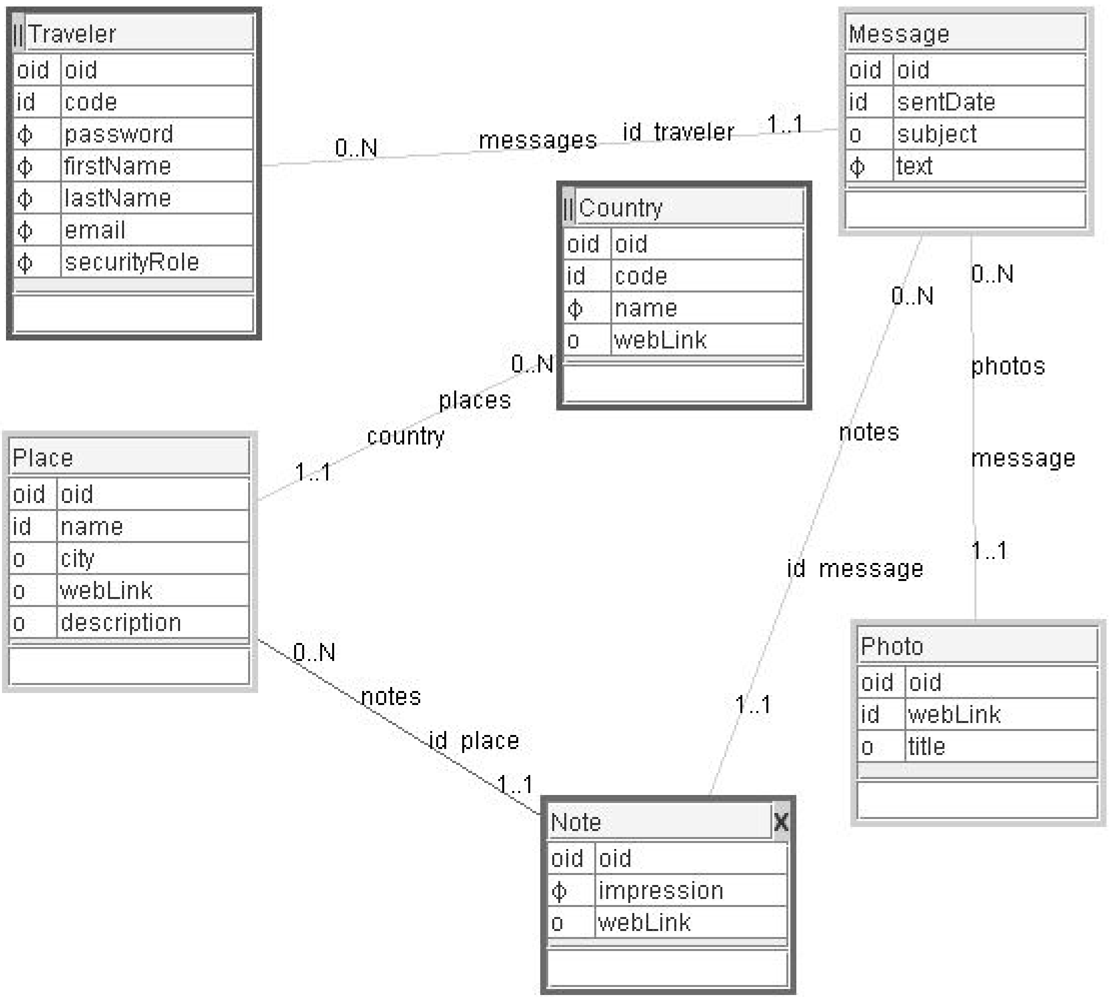


Figure 3.7 Travel Impression: Spiral 03

There are two entry concepts that allow a user to enter the model's data through countries or travelers. The two concepts have the entry sign in the left section of their title areas. There will be one XML data file for each entry concept. For example, when a country is selected, its places may be "visited". For each place, its impression notes made by various travelers can be seen. The relationship between the Country and Place concepts is internal, which is indicated by a lighter line in ModelibraModeler. This means that places will be saved within their country. The relationship between the Place and Note concepts is external. This means that notes about a place are not saved within that place. They are saved with their message parent. An external line is obtained from an internal line by using a pop-up menu on the line and un-selecting the *Internal?* check box.

The Note concept is an intersection concept between the Place and Message concepts. This means that semantically the relationship between the Place and Message concepts is of the many-to-many type. A message may mention many places, and a place may be "noted" by many messages. In a pop-up menu on each participating line, the *Part of Many-to-many?* check box must be selected to obtain the X sign in the right section of the title area.

If there is a need for more web links for the same place, another concept could be introduced. This is done in Figure 3.8 by connecting the Url concept with the Place concept. Also, the note property is added to both Traveler and Photo concepts.

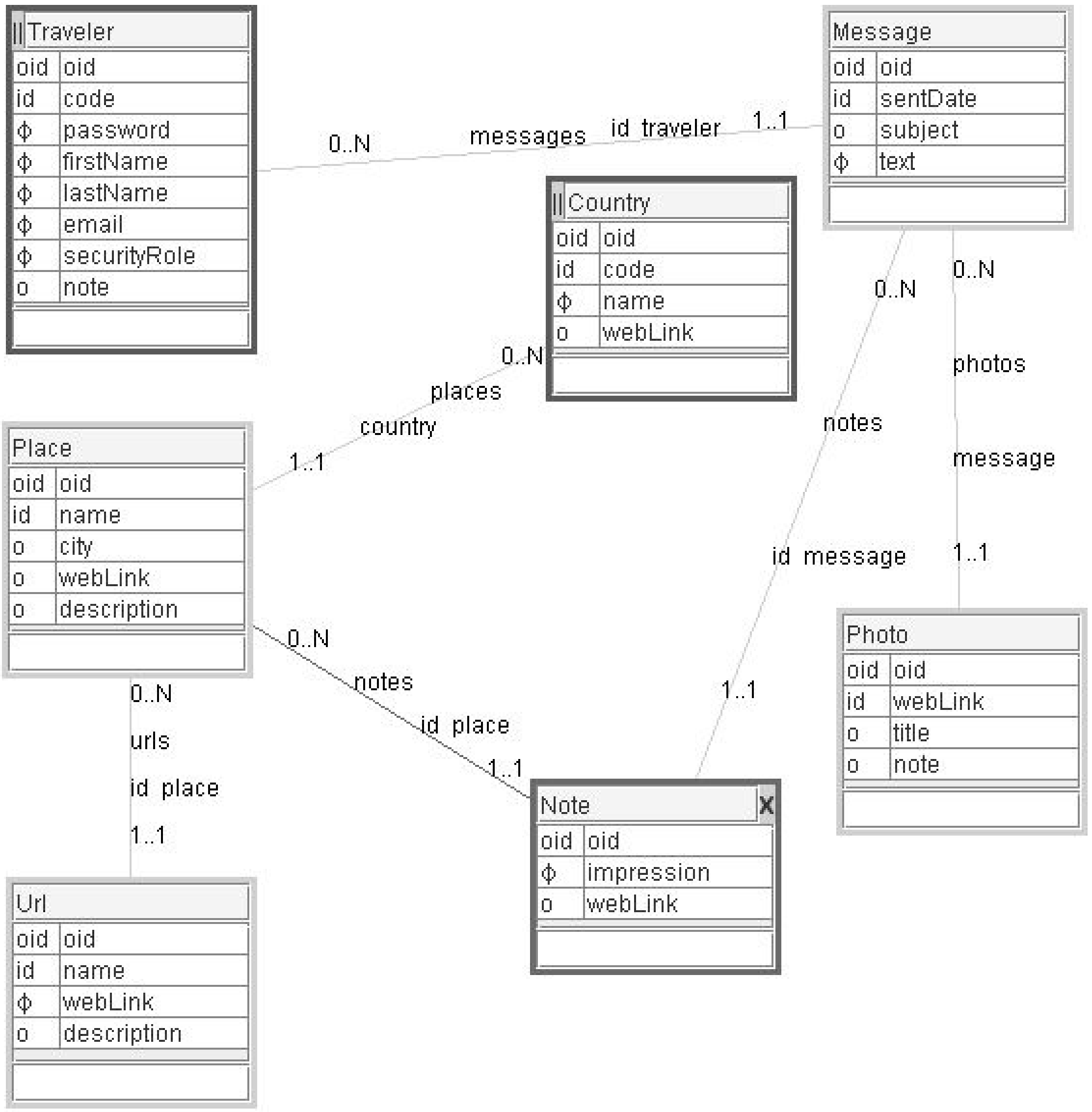


Figure 3.8 Travel Impression: Spiral 04

The model could be further extended with more concepts, properties and relationships. It is a good practice to spend more time on the model by introducing additional spirals. The new spirals may clarify some issues and provide new ideas. However, it is also a good practice to start programming with a less ambitious model, but with the knowledge of a richer model.

All model spirals are in the TravelImpression.mm file that can be opened by ModelibraModeler. If you cannot open the file, import first the types from the TravelImpression.type file and then diagrams from the TravelImpression.diagram file. All three files are in the mm/TravelImpression directory of the DmEduc-01 project. The same files can be found in the mm/spiral directory of the TravelImpression-00 project at JavaForge. The TravelImpression-00 project is created from the ModelibraWicketSkeleton project.

The ModelibraWicketSkeleton project is first checked out from JavaForge. This is done only once and in future the project will be synchronized with the server version for local updates. The local version of the ModelibraWicketSkeleton project is then exported by using the *File/Export.../General/File System* action path in Eclipse, but without the css, template, classes, lib and logs directories, and with the *Create directory structure for files* option. The whole project is copied to another location without the directories and files created by Subversion. The copied project is renamed, before the project is imported into Eclipse by *File/Import.../General/Existing Projects into a Workspace*. The new project is committed to the Subversion repository by the *Team/Share project.../SVN* pop-up menu item. A list of files to share is presented and not all of them are selected to be transferred to the repository. For example, the project classes and logs are not selected.

In a real project only the last version of the project is kept in the trunk directory of a repository. For this last version changes are regularly committed. Previous versions (or spirals) are kept in the tags directory and they are not modified. Thus, when a new project is put for the first time in an SVN repository, this is accomplished by *Team/Share project.../SVN*. When a certain state of the project is declared to be a specific version, this is done by *Team/Branch/Tag...* . The project in the trunk directory (from) is tagged with the new name in the tags directory (to). This new name is usually the name of the project concatenated with the version or spiral number.

Both DmEduc and TravelImression projects are created for educational purposes. Both projects have a sequence of spirals that are kept in the trunk directory. They are kept in the trunk directory since they are changed from time to time to follow the evolution of Modelibra.

In order to prevent transferring of classes and logs in future commits, the svn:ignore property is defined at the WEB-INF directory for the classes and logs directory. This is done by selecting the WEB-INF directory and using the *Team/Set Property...* pop-up menu item.

Since the new project does not have the content for the css, template and lib directories, the svn:externals property is defined at the WEB-INF directory for the lib directory, and at the project's root directory for the css and template directories. Note that the external SVN repository at JavaForge is used to obtain the latest version of jar files, css definitions and code generation templates from the source, which is the ModelibraWicketSkeleton project. In this way, the Update of the local project, which is externally linked to Modelibra at JavaForge, will bring the latest changes of those files.

The following are detailed instructions how to start a new ModelibraWicket project, from the doc/start.odt file in the ModelibraWicketSkeleton project.

How to start a new ModelibraWicket project:

export the ModelibraWicketSkeleton project without the css, template, classes, lib and logs directories to a new location outside the Modelibra workspace: *File/Export.../General/File System* (use the *Create directory structure for files* option)

rename the project directory to reflect your own project

rename the Eclipse project to your own project using an ordinary text editor

import the project into Eclipse

select the project and use the pop-up menu: *Team/Share Project...* (use the *SVN repository*; share the project under the trunk directory) do not share the empty classes directory

define by *Team/Set Property...* the svn:ignore property at the WEB-INF directory for the classes and logs directories

define the svn:externals property at the WEB-INF directory for the lib directory:
lib http://svn.javaforge.com/svn/Modelibra/trunk/ModelibraWicketSkeleton/WEB-INF/lib

commit by *Team/Commit...* local changes to the repository

update by *Team/Update* changes from the repository to the local version

define the svn:externals property at the project's root directory for the css and template directories:
css http://svn.javaforge.com/svn/Modelibra/trunk/ModelibraWicketCss/css
template http://svn.javaforge.com/svn/Modelibra/trunk/ModelibraWicketSkeleton/template

commit by *Team/Commit...* local changes to the repository

update by *Team/Update* changes from the repository to the local version

design a domain model in ModelibraModeler and save it in the mm directory

generate the reusable-model-config.xml file (use the minimal configuration) from ModelibraModeler into the WEB-INF/config directory

refresh the project to examine the reusable-model-config.xml file (improve the format if necessary)

in the dm.gen.DmGenerator class change the DOMAIN_CODE constant to your domain code (if not sure, check the reusable-model-config.xml file)

run the main method of the DmGenerator class as Java application

refresh the project to see the generated code, data and web xml configuration

in the generated Java classes use *Ctrl-Shift O F S* to clean imports, format the code and save it

if a concept was referenced as if it was an entry point, add the specific get method to the specific model class

commit by *Team/Commit...* local changes to the repository (*Select* all unversioned files)

update by *Team/Update* changes from the repository to the local version

run the main method of the Start class to start the Jetty server

check the WEB-INF/logs/info.html directory for info and error messages

use the ModelibraWicket application in your browser: http://localhost:8081/

use the doc/spiral.txt file to add your project notes

The new project is the base for the next spiral where both Modelibra and ModelibraWicket will be used and the code for the domain model and its views will be generated.

Follow a similar type of reasoning as in this section and create multiple model spirals for the domain of your choice. If you are a student in a course, form a team of two to three members and start developing a new web application by using Modelibra and Wicket.

Summary

The domain model from the previous spiral has only one concept. The new version of the domain model in this chapter has still only one concept, but additional properties are added. A way how to add properties and define their types in ModelibraModeler is explained. The new version of the domain configuration is presented.

In Modelibra, neither the domain nor the model has any knowledge about the persistence of data. However, there is a trivial new class in the domain package to handle the persistence of entities. The concept classes are augmented with new properties and new methods. A serious of tests is done in the only concept test class. Those methods from Modelibra interfaces used in this chapter are indicated.

The spiral approach to designing a domain model for the new project is practiced. The new project is started at the model design level to show how Modelibra is applied in the development process.

The next chapter will start the new section of the book. The section is about views of the domain model. The domain model will become alive as a web application. The existing data will be displayed as web pages and the new data will be created through web forms.

Questions

1. How to create new properties for a concept in ModelibraModeler without using the lower area of the concept?

2. Why is it important to export types and diagrams in ModelibraModeler?

3. How to create a new model spiral in ModelibraModeler?

4. How do you know where the model data will be saved in Modelibra?

5. What should you change to allow for the explicit use of the load and save data methods in Modelibra?

6. Find a method in Modelibra interfaces that has not yet been used and explain its purpose.

Exercises

Exercise 3.1.

Repeat the same tests without removing urls created in tests. Consult the log to see what happens. You can always empty the data file in Eclipse by keeping only the urls XML element.

```
<?xml version="1.0" encoding="UTF-8"?>

<urls>

</urls>
```

Exercise 3.2.

Create at least one new test at the concept level.

Exercise 3.3.

Redo the tests in this spiral in a new test class as regular private methods and call them one by one in the main method of the test class.

Exercise 3.4.

Find a method in Modelibra interfaces that has not yet been used and create a new test with that method

Exercise 3.5.

Create an Eclipse project with a new model in a new domain. The model should have at least one concept that is different from the Url concept.

Web Links

[Flickr] Online Photo Management
http://www.flickr.com/

[Overloading] Method Overloading
http://java.sys-con.com/read/38068.htm