Chapter 10: Web Application

The objective of this chapter is to provide a user sign in and to support international versions of the web application. In ModelibraWicket, by default, there is no need to sign in to display data. However, for updates the sign in is mandatory if the domain is configured to use the sign in. In addition, the configuration must state what concept is used for the sign in. In this spiral, the Member concept provides the code and the password for the sign in. If the application is configured to support internationalization (I18n), the CountryLanguage concept must be present in the application domain.

The home page of the web application has three new web sections. The *Sign In* section with user code and password form fields is displayed at the beginning of the right column of the home page, The *I18n* section is a drop down choice that is a part of the menu of the home page. A user may change the language from English to French or Bosnian. English is the default language. The *Countries* section is located at the end of the right column of the home page. The section component is a table with country codes and names. Since there are many countries, the component provides the block (of rows) bar so that only a small number of countries may be seen at once. This number can be changed in the domain configuration.

The web application has several new pages: *About*, *FAQ*, *Contact Us* and *Sign Up*. The *About* page displays two domain models and provides a list of web links for the Links category. From the menu of the *About* page a user may go to the *FAQ* or *Contact Us* page. The *FAQ* page lists questions and responses for the FAQ category. The *Contact Us* page provides a form for a user comment or request together with a source of the comment, which can be a name, an email, a web address, or something else. The *Sign Up* page allows a new user to register as a regular member. The provided code and password values may be used later to sign in as a regular member with all rights and restrictions of the regular security role. The other security roles are: admin, manager, advanced and casual. The admin role does not have any restrictions, while the casual role has many. The regular role is positioned between the advanced and casual roles.

Domain Model

The Web Link model from the previous spiral is used without changes in this chapter (Figure 10.1).

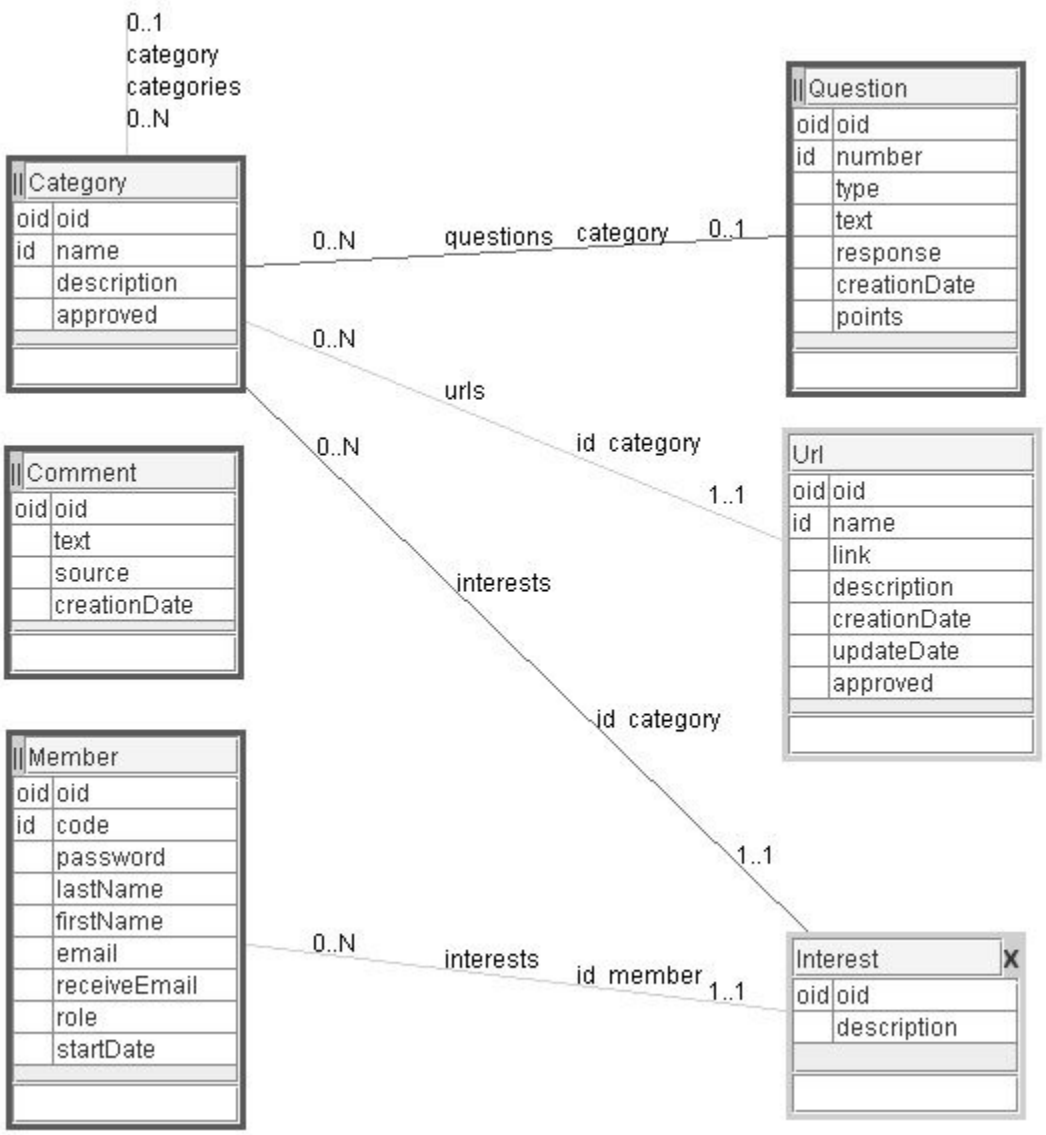

Figure 10.1. Web Link Model

However, there is a new model in the application domain. The model is named Reference because it is used for reference purposes (Figure 10.2). It has three concepts: CountryName, CountryLanguage and SecurityRole. Only the last two concepts are essential for the application. The country languages are used in the I18n component, and the security roles are used to determine the rights of the signed in member.

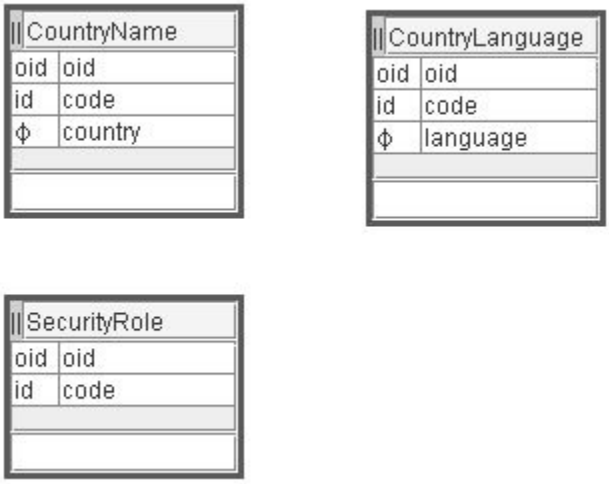

Figure 10.2. Reference Model

Domain Configuration

The DmEduc domain is configured to use the Reference model to look for reference data if they are not present in the principal model. The Reference model could have been named differently. The domain, and as a consequence the domain's web application, has i18n and signin elements set to **true**. This is done in the specific domain configuration. The sign in concept is Member. In tables, only a small portion of a longer text is displayed. The default value is changed by the new value of 48 characters. If a table has many entities, only a small block of entities is displayed at once. The new value is set to 16 entities.

```
<domain oid="1189989374359">
      <code>DmEduc</code>
      <type>Specific</type>

      <referenceModel>Reference</referenceModel>

      <i18n>true</i18n>
      <signin>true</signin>
      <signinConcept>Member</signinConcept>
      <shortTextDefaultLength>48</shortTextDefaultLength>
      <pageBlockDefaultSize>16</pageBlockDefaultSize>
```

The Reference model has three concepts: CountryLanguage, CountryName and SecurityRole. The reusable configuration is generated by ModelibraModeler.

```
<domains>

      <domain oid="1189989374359">
            <code>DmEduc</code>
            <type>Reusable</type>

            <models>
```

```
...

<model oid="1195157999719">
      <code>Reference</code>
      <author>Dzenan Ridjanovic</author>
      <persistenceType>xml</persistenceType>
      <persistenceRelativePath>
            data/xml/dmeduc/reference
      </persistenceRelativePath>
      <defaultLoadSave>true</defaultLoadSave>

      <concepts>

            <concept oid="1176319565377">
                  <code>CountryName</code>
                  <entitiesCode>CountryNames</entitiesCode>
                  <entry>true</entry>

                  <properties>
                        <property oid="1176319583347">
                              <code>code</code>
                              <propertyClass>
                                    java.lang.String
                              </propertyClass>
                              <maxLength>16</maxLength>
                              <required>true</required>
                              <unique>true</unique>

                              <essential>true</essential>
                        </property>
                        <property oid="1176319586239">
                              <code>country</code>
                              <propertyClass>
                                    java.lang.String
                              </propertyClass>
                              <maxLength>64</maxLength>
                              <required>true</required>

                              <essential>true</essential>
                        </property>
                  </properties>

                  <neighbors>
                  </neighbors>

            </concept>

            <concept oid="1176319588568">
                  <code>CountryLanguage</code>
                  <entitiesCode>CountryLanguages</entitiesCode>
                  <entry>true</entry>
```

```
        <properties>
            <property oid="1176319598351">
                <code>code</code>
                <propertyClass>admin
                    java.lang.String
                </propertyClass>
                <maxLength>16</maxLength>
                <required>true</required>
                <unique>true</unique>

                <essential>true</essential>
            </property>
            <property oid="1176319601087">
                <code>language</code>
                <propertyClass>
                    java.lang.String
                </propertyClass>
                <maxLength>32</maxLength>
                <required>true</required>

                <essential>true</essential>
            </property>
        </properties>

        <neighbors>
        </neighbors>

    </concept>

    <concept oid="1176319603400">
        <code>SecurityRole</code>
        <entitiesCode>SecurityRoles</entitiesCode>
        <entry>true</entry>

        <properties>
            <property oid="1176319611762">
                <code>code</code>
                <propertyClass>
                    java.lang.String
                </propertyClass>
                <maxLength>16</maxLength>
                <required>true</required>
                <unique>true</unique>

                <essential>true</essential>
            </property>
        </properties>

        <neighbors>
        </neighbors>
```

```
                    </concept>

                </concepts>

            </model>

        </models>

    </domain>

</domains>
```

A country language can be added, but not removed. The same is true for a country name. A new security role cannot be even added. All this is done in the specific configuration for the Reference model.

```
            <model oid="1195157999719">
                <code>Reference</code>
                <extension>true</extension>
                <extensionDomain>DmEduc</extensionDomain>
                <extensionDomainType>Reusable</extensionDomainType>
                <extensionModel>Reference</extensionModel>

                <concepts>

                    <concept oid="1176319565377">
                        <code>CountryName</code>
                        <extension>true</extension>
                        <extensionConcept>CountryName</extensionConcept>

                        <displayType>list</displayType>
                        <add>true</add>
                        <remove>false</remove>
                        <update>false</update>

                        <properties>

                            <property oid="1176319583347">
                                <code>code</code>
                                <extension>true</extension>
                                <extensionProperty>code</extensionProperty>

                                <maxLength>2</maxLength>

                            </property>

                            <property oid="1176319586239">
                                <code>country</code>
                                <extension>true</extension>
                                <extensionProperty>
                                    country
                                </extensionProperty>
```

```
            </property>

      </properties>

</concept>

<concept oid="1176319588568">
      <code>CountryLanguage</code>
      <extension>true</extension>
      <extensionConcept>CountryLanguage</extensionConcept>

      <displayType>list</displayType>
      <add>true</add>
      <remove>false</remove>
      <update>false</update>

      <properties>

            <property oid="1176319598351">
                  <code>code</code>
                  <extension>true</extension>
                  <extensionProperty>code</extensionProperty>

                  <maxLength>2</maxLength>

            </property>

            <property oid="1176319601087">
                  <code>language</code>
                  <extension>true</extension>
                  <extensionProperty>
                        language
                  </extensionProperty>

            </property>

      </properties>

</concept>

<concept oid="1176319603400">
      <code>SecurityRole</code>
      <extension>true</extension>
      <extensionConcept>SecurityRole</extensionConcept>

      <min>1</min>
      <max>5</max>

      <displayType>list</displayType>
      <add>false</add>
      <remove>false</remove>
```

```
                    <update>false</update>

                    <properties>

                        <property oid="1176319611762">
                            <code>code</code>
                            <extension>true</extension>
                            <extensionProperty>code</extensionProperty>

                        </property>

                    </properties>

                </concept>

            </concepts>

        </model>
```

The Member entry concept has the code and password properties that are used in the Sign In web component. The security of the application is managed through member roles. The role property in the Member concept has the new configuration.

```
<property oid="1172170850713">
    <code>role</code>
    <extension>true</extension>
    <extensionProperty>role</extensionProperty>

    <validateType>true</validateType>
    <validationType>
        SecurityRoles
    </validationType>

    <update>false</update>
</property>
```

The property is validated by Modelibra using the entities with the SecurityRoles name. A value in the role property is valid if it is one of code values found in SecurityRoles. In ModelibraWicket, for a property with the type validation based on entities, a drop down choice is presented to a user. In this way only a valid value can be used. However, the type validation is done in Modelibra anyway, since Modelibra does not have any knowledge of ModelibraWicket.

The securityrole.xml data file has five values. The five codes are predefined by ModelibraWicket and should not be changed.

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

<securityRoles>
  <securityRole oid="1160065956256">
    <code>admin</code>
  </securityRole>
```

```
  <securityRole oid="1160065956266">
    <code>manager</code>
  </securityRole>
  <securityRole oid="1160065956270">
    <code>advanced</code>
  </securityRole>
  <securityRole oid="1160065956276">
    <code>regular</code>
  </securityRole>
  <securityRole oid="1160065956279">
    <code>casual</code>
  </securityRole>
</securityRoles>
```

The security roles determine if users have display and update rights for concepts, properties and neighbors. The display of concepts, properties and neighbors may be done without sign in. Without sign in, all users have the same display rights determined by the display configurations of concepts, properties and neighbors. With sign in, display and update rights are determined by the security roles.

Table 10.1. Security roles

	display	**add**	**update**	**remove**
admin	true	true	true	true
manager	true	true	true	config
advanced	true	config	config	config
regular	config	config	config	config
casual	config	config	config	false

The admin, manager and advanced roles have all display rights (true even when false in display configurations of concepts, properties and neighbors). The regular and casual roles have display rights only for true display configurations of concepts, properties and neighbors.

The admin and manager roles have all add rights (true even when false in add configurations of concepts). The advanced, regular and casual roles have add rights only for true add configurations of concepts.

The admin and manager roles have all update rights (true even when false in update configurations of properties and neighbors). The advanced, regular and casual roles have update rights only for true update configurations of properties and neighbors.

The admin role has all remove rights (true even when false in remove configurations of concepts). The manager, advanced and regular roles have remove rights only for true remove configurations of concepts. The casual role does not have remove rights (false even when true in remove configurations of concepts).

The Member concept allows the add and update actions, while it restricts the remove action. In addition, since the Member concept is used in sign in, a regular member can update only her own record.

```
<concept oid="1172170727628">
    <code>Member</code>
```

```
<extension>true</extension>
<extensionConcept>Member</extensionConcept>

<add>true</add>
<remove>false</remove>
<update>true</update>
```

Code Generation

The Reference model is added to the DmEduc.mm file and the reusable domain configuration is regenerated.

The DmGenerator class is used to generate code for the Reference model. First, the domain Gen classes are generated. Then, all classes and data files for the Reference model are generated. The backup of the specific domain configuration is made. Then the specific domain configuration is generated and the two specific domain configurations are carefully merged into the new version of the specific domain configuration. For the Wicket part, page classes for all concepts are generated for the Reference model. Finally, the application properties for both models are regenerated in the DmEducApp.properties file.

```
// *** 2. Generate new with preserving specific ***
// *** All for an additional model
dmGenerator.getDmModelibraGenerator().generateDomainGenClass();
dmGenerator.getDmModelibraGenerator().generateModel("Reference");
// *** Optional: If done, be sure to have a backup of
// specific-domain-config.xml ***
dmGenerator.getDmModelibraGenerator().generateSpecificDomainConfig();

// *** 2. Generate new with preserving specific ***
dmGenerator.getDmModelibraWicketGenerator()
      .generateConceptPageClasses("Reference");
// *** All generic properties ***
dmGenerator.getDmModelibraWicketGenerator()
      .generateModelibraWicketAppProperties();
```

The web application is used and the data for the Reference model are entered. Once entered, the data can be reused in another web application by coppying the corresponding XML data files.

Home Page

The HomePage class has two new components: EntityDisplayTablePanel for country codes and names, and SigninPanel for validating a user. Both web components come from ModelibraWicket.

Country codes and names are displayed in a table similar to the web component that displays comments. The title of the table is determined by the CountryNames key. When a language is changed, the value for that key and for that language is displayed.

If the domain is configured not to use sign in, the SigninPanel component is not visible. Otherwise, if a

user is not signed in, the SigninPanel component is visible. When a user signs in, the component becomes invisible. If a user makes a mistake, the sign in feedback section will show the error message.

```
package dmeduc.wicket.app.home;

import org.apache.wicket.markup.html.panel.FeedbackPanel;
import org.apache.wicket.markup.html.panel.Panel;
import org.modelibra.wicket.concept.EntityDisplayTablePanel;
import org.modelibra.wicket.concept.EntityPropertyDisplayListPanel;
import org.modelibra.wicket.container.DmPage;
import org.modelibra.wicket.neighbor.ParentChildPropertyDisplayListPanel;
import org.modelibra.wicket.security.SigninPanel;
import org.modelibra.wicket.view.View;
import org.modelibra.wicket.view.ViewModel;

import dmeduc.DmEduc;
import dmeduc.reference.Reference;
import dmeduc.reference.countryname.CountryNames;
import dmeduc.weblink.WebLink;
import dmeduc.weblink.category.Categories;
import dmeduc.weblink.comment.Comments;
import dmeduc.weblink.member.Member;
import dmeduc.weblink.member.Members;
import dmeduc.weblink.question.Questions;
import dmeduc.wicket.app.DmEducApp;
import dmeduc.wicket.weblink.member.MemberInterestsListPanel;

public class HomePage extends DmPage {

	public HomePage() {
		DmEducApp dmEducApp = (DmEducApp) getApplication();
		DmEduc dmEduc = dmEducApp.getDmEduc();
		WebLink webLink = dmEduc.getWebLink();
		Reference reference = dmEduc.getReference();

		ViewModel homePageModel = new ViewModel(webLink);
		View homePageView = new View(dmEducApp);
		homePageView.setWicketId("homeMenu");
		homePageView.setPage(this);

		// Menu
		add(new HomeMenu(homePageModel, homePageView));

		...

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

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

```
            commentsView.setContextView(homePageView);

            EntityDisplayTablePanel commentTable = new EntityDisplayTablePanel(
                  commentsModel, commentsView);
            add(commentTable);

            // Country Names
            ViewModel countryNamesViewModel = new ViewModel(reference);
            CountryNames countryNames = reference.getCountryNames();
            countryNamesViewModel.setEntities(countryNames);

            View countryNamesView = new View(dmEducApp);
            countryNamesView.setWicketId("countryNameTable");
            countryNamesView.setContextView(homePageView);

            EntityDisplayTablePanel countryTable = new EntityDisplayTablePanel(
                  countryNamesViewModel, countryNamesView);
            add(countryTable);

            // Sign In
            ViewModel signInViewModel = new ViewModel(webLink);
            signInViewModel.setEntities(members);
            signInViewModel.setEntity(new Member(members.getModel()));

            View signInView = new View(dmEducApp);
            signInView.setWicketId("signIn");
            signInView.setContextView(homePageView);

            Panel signIn = new SigninPanel(signInViewModel, signInView);
            add(signIn);
            add(new FeedbackPanel("signInFeedback"));
            if (!dmEduc.getDomainConfig().isSignin()) {
                  signIn.setVisible(false);
            } else if (getAppSession().isUserSignedIn()) {
                  signIn.setVisible(false);
            }
      }

}
```

The view model of the SigninPanel component provides, for the sake of flexibility, the possibility to use different properties for code and password. The only restriction is that the two new properties are of the String type and that the sign in concept is an entry concept that has those two properties. The new property names can be passed to the view model of the component as user properties.

```
      countryNamesViewModel.getUserProperties().
            addUserProperty("code", "differentCodePropertyName");
      countryNamesViewModel.getUserProperties().
            addUserProperty("password", "differentPasswordPropertyName");
```

The side bar in the HomePage.html file has new sections.

```
<div class="sidebar">
      <div wicket:id="signInn">
            To be replaced dynamically by the Sign In form.
      </div>
      <div wicket:id = "signInFeedback"/>
      <br/>
      <div wicket:id="commentTable">SecurityRoles
            To be replaced dynamically by the table of comments.
      </div>
      <br/>
      <div wicket:id="questionTextList">
            To be replaced dynamically by the list of questions.
      </div>
      <br/>
      <div wicket:id="memberInterestList">
            To be replaced dynamically by the list of members
            each with his/her interests.
      </div>
      <div wicket:id="countryNameTable">
            To be replaced dynamically by the table of countries.
      </div>
      <br/>
</div>
```

Home Menu

The HomeMenu class has four new components: a link to sign up, a link to sign out, a drop down choice of natural languages, and a link to the About page, which is defined in the new dmeduc.wicket.about package.

The sign out link is visible only if a user is signed in. If a user decides to sign out, the corresponding session is invalidated and the home page is displayed again but this time with the sign in component.

```
package dmeduc.wicket.app.home;

import org.apache.wicket.markup.html.link.Link;
import org.apache.wicket.markup.html.link.PageLink;
import org.apache.wicket.markup.html.panel.Panel;
import org.modelibra.wicket.app.domain.DomainPage;
import org.modelibra.wicket.concept.CountryLanguageChoicePanel;
import org.modelibra.wicket.container.DmPanel;
import org.modelibra.wicket.view.View;
import org.modelibra.wicket.view.ViewModel;

import dmeduc.DmEduc;
import dmeduc.reference.Reference;
import dmeduc.reference.countrylanguage.CountryLanguage;
import dmeduc.reference.countrylanguage.CountryLanguages;
import dmeduc.weblink.WebLink;
```

```
import dmeduc.weblink.member.Member;
import dmeduc.weblink.member.Members;
import dmeduc.wicket.app.DmEducApp;
import dmeduc.wicket.app.about.AboutPage;
import dmeduc.wicket.weblink.member.SignUpPage;

public class HomeMenu extends DmPanel {

	public HomeMenu(final ViewModel viewModel, final View view) {
		super(view.getWicketId());
		DmEducApp dmEducApp = (DmEducApp) getApplication();
		DmEduc dmEduc = dmEducApp.getDmEduc();
		WebLink webLink = dmEduc.getWebLink();
		Reference reference = dmEduc.getReference();

		// Sign Up
		ViewModel signUpViewModel = new ViewModel(webLink);
		Members members = webLink.getMembers();
		signUpViewModel.setEntities(members);
		signUpViewModel.setEntity(new Member(members.getModel()));

		PageLink signUpLink = SignUpPage.link("signUpLink",
			signUpViewModel, view);
		add(signUpLink);
		if (!dmEduc.getDomainConfig().isSignin()) {
			signUpLink.setVisible(false);
		} else if (getAppSession().isUserSignedIn()) {
			signUpLink.setVisible(false);
		}

		// Sign Out
		Link signOutLink = new Link("signOutLink") {
			public void onClick() {
				getAppSession().signOutUser();
				setResponsePage(HomePage.class);
			}
		};
		add(signOutLink);
		if (!getAppSession().isUserSignedIn()) {
			signOutLink.setVisible(false);
		}

		// Domain
		Link domainLink = new Link("domainLink") {
			public void onClick() {
				setResponsePage(new DomainPage());
			}
		};
		add(domainLink);

		// About
		Link aboutLink = new Link("aboutLink") {
```

```
            public void onClick() {
                setResponsePage(new AboutPage(viewModel, view));
            }
        };
        add(aboutLink);

        // I18n
        ViewModel countryLanguageViewModel = new ViewModel(reference);
        CountryLanguages countryLanguages = reference.getCountryLanguages();
        countryLanguageViewModel.setEntities(countryLanguages);
        String languageCode = null;
        CountryLanguage defaultLanguage = null;
        languageCode = getAppSession().getLocale().getLanguage();
        defaultLanguage = countryLanguages.getCountryLanguage(languageCode);
        if (defaultLanguage == null) {
            defaultLanguage = countryLanguages.getCountryLanguage("en");
        }
        countryLanguageViewModel.setEntity(defaultLanguage);

        View countryLanguageView = new View(dmEducApp);
        countryLanguageView.setWicketId("countryLanguageChoice");

        Panel countryLanguageChoice = new CountryLanguageChoicePanel(
            countryLanguageViewModel, countryLanguageView);
        add(countryLanguageChoice);
        if (!dmEduc.getDomainConfig().isI18n()) {
            countryLanguageChoice.setVisible(false);
        }
    }

}
```

The sign up link is used to register a new user as a regular member. The sign up link is visible only if a user is not signed. A new Member entity is created to accept new values. The Sign Up page is not constructed immediately. The static link method of the SignUpPage class is used to postpone the creation of the page until a user clicks on the link in the web application.

```
    public static PageLink link(final String linkId, final ViewModel viewModel,
            final View view) {
        PageLink link = new PageLink(linkId, new IPageLink() {
            public Page getPage() {
                return new SignUpPage(viewModel, view);
            }

            public Class<? extends Page> getPageIdentity() {
                return SignUpPage.class;
            }
        });
        return link;
    }
```

This is done to allow a member to sign in first, then to send a comment. However, a user that is not a member may also contact us. The following is extracted from the JavaDoc of Wicket [Wicket JavaDoc] for the IPageLink interface from the org.apache.wicket.markup.html.link package:

> "Interface that is used to implement delayed page linking. The getPage() method returns an instance of Page when a link is actually clicked (thus avoiding the need to create a destination Page object for every link on a given Page in advance)."

When the home page is displayed, the default language is the language of the session. However, if the session does not have the language, the English language becomes the default language.

The view model of the language component provides, for the sake of flexibility, the possibility to add names of three methods as user properties. This allows a different class name for the natural languages with different methods. The following three methods can be renamed differently and passed to the web component as user properties. However, if they are not renamed there is no need to pass them to the web component.

```
countryLanguageViewModel.getUserProperties().
      addUserProperty("getLanguageMethod", "getLanguage");
countryLanguageViewModel.getUserProperties().
      addUserProperty("getLanguageListMethod", "getLanguageList");
countryLanguageViewModel.getUserProperties().
      addUserProperty("getLanguageCodeMethod", "getLanguageCode");
```

The getLanguage method is located in the GenCountryLanguage class.

```
public String getLanguage() {
      return language;
}
```

The getLanguageCode and getLanguageList specific methods are added to the CountryLanguages class.

```
public String getLanguageCode(String language) {
      String languageCode = null;
      CountryLanguage countryLanguage = getCountryLanguage("language",
                  language);
      if (countryLanguage != null) {
            languageCode = countryLanguage.getCode();
      }
      return languageCode;
}

public List<String> getLanguageList() {
      List<String> languageList = new ArrayList<String>();
      for (CountryLanguage countryLanguage : this) {
            languageList.add(countryLanguage.getLanguage());
      }
      return languageList;
```

```
    }
```

The three methods may have different names but they must have the same argument and return types.

There is one more specific method in the CountryLanguages class, added for convenience reasons only. Its name is getCountryLanguage and it has only one argument reserved for the language code.

```
    public CountryLanguage getCountryLanguage(String languageCode) {
        return retrieveByCode(languageCode);
    }
```

The DmEducApp class has now four different files with the properties extension: DmEducApp and DmEducApp_en for English (default), DmEducApp_fr for French and DmEducApp_ba for Bosnian. The DmEducApp.properties file is used in the code generation and its content should not be changed. The DmEducApp_en.properties file is used to specialize text in English. The other two files have the same keys on the left and their own key values on the right of the equal sign. Properties at the level of the web application describe both the domain and the models of the application.

Specific page classes may have their own properties. In the spirit of the reuse of components, the keys used in a panel component should be kept in the properties file that carries the same name as the name of the component. For example, the four keys that appear in the HomeMenu.html file, should be placed in the HomeMenu.propereties file. In this case, if a key is not found at the panel level, it is searched at the page level, and finally at the application level.

Thus, HomeMenu has the specific support for I18n. Its keys, which appear in the HomeMenu component, are kept in the HomePage.properties file for English,

```
home.menu.about=About
home.menu.admin=Administration
home.menu.home=Home
home.menu.signOut=Sign Out
home.menu.signUp=Sign Up
```

and in the HomeMenu_fr.properties file for French.

```
home.menu.about=À propos
home.menu.admin=Administration
home.menu.home=Page principale
home.menu.signOut=Fermeture de session
home.menu.signOut=Enregistrement
```

Only the keys, and not the key values, are used in the HTML code. For example, the HomeMenu.html file refers to the keys by the wicket:message element.

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

<html xmlns:wicket>

<wicket:panel>
```

```
<div class="menu">
      <ul>
            <li>
                  <a href="http://www.modelibra.org/">
                        <wicket:message key = "home.menu.home"/>
                  </a>
            </li>
            <li>
                  <a wicket:id="signUpLink">
                        <wicket:message key = "home.menu.signUp"/>
                  </a>
            </li>
            <li>
                  <a wicket:id="signOutLink">
                        <wicket:message key = "home.menu.signOut"/>
                  </a>
            </li>
            <li>
                  <a wicket:id="domainLink">
                        <wicket:message key = "home.menu.admin"/>
                  </a>
            </li>
            <li>
                  <a wicket:id="aboutLink">
                        <wicket:message key = "home.menu.about"/>
                  </a>
            </li>
             
             
            <span wicket:id="countryLanguageChoice">
                  To be replaced dynamically by the language choice.
            </span>
      </ul>
</div>

</wicket:panel>

</html>
```

About Page

The About page displays the two models of the DmEduc domain as two image files. This is done statically only in the corresponding About HTML page. Both models are displayed, one after another, in the left column of the About page. In the right column, there is a list of web links for the category with the Links name. The About page has also a menu that lets a user return to the context of the page.

```
package dmeduc.wicket.app.about;

import org.apache.wicket.markup.html.panel.Panel;
import org.modelibra.wicket.concept.EntityPropertyDisplayListPanel;
```

```
import org.modelibra.wicket.container.DmPage;
import org.modelibra.wicket.view.View;
import org.modelibra.wicket.view.ViewModel;

import dmeduc.DmEduc;
import dmeduc.weblink.WebLink;
import dmeduc.weblink.category.Categories;
import dmeduc.weblink.category.Category;
import dmeduc.weblink.url.Urls;
import dmeduc.wicket.app.DmEducApp;

public class AboutPage extends DmPage {

	public AboutPage(final ViewModel viewModel, final View view) {
		DmEducApp dmEducApp = (DmEducApp) getApplication();
		DmEduc dmEduc = dmEducApp.getDmEduc();
		WebLink webLink = dmEduc.getWebLink();

		// Menu
		View menuView = new View(dmEducApp);
		menuView.setWicketId("aboutMenu");
		menuView.setPage(this);
		menuView.setContextView(view);

		add(new AboutMenu(viewModel, menuView));

		// Links Category Urls
		ViewModel linksModel = new ViewModel(webLink);
		Categories categories = webLink.getCategories();
		Category linksCategory = categories.getCategoryByName("Links");
		if (linksCategory != null) {
			Urls linksCategoryUrls = linksCategory.getUrls()
					.getApprovedUrls();
			linksModel.setEntities(linksCategoryUrls);
			linksModel.setPropertyCode("link");
		}

		View linksView = new View(dmEducApp);
		linksView.setPage(this);
		linksView.setWicketId("linksCategoryUrlsList");

		Panel links;
		if (linksCategory != null) {
			links = new EntityPropertyDisplayListPanel(linksModel, linksView);
		} else {
			links = new Panel("linksCategoryUrlsList");
			links.setVisible(false);
		}
		add(links);
	}

}
```

The AboutPage class has a constructor with the ViewModel and View arguments. The AboutMenu class is used for the page menu. The View argument from the constructor of the AboutPage class is used to define the context for the menu. Note that only the menu view is created. The menu model is reused from the context.

The About page displays web links for the Links category. If this category is found, its urls are obtained and only a subset of approved urls is used. If the category is not found, the web component is made invisible. In Wicket, there must be a strict correspondence between the HTML element with a Wicket id and the Java web component with the same Wicket id. If the web component cannot be properly created, it must be constructed and then made invisible. This the reason that the Panel class from Wicket is used when the Links category is not found.

The body of the AboutPage.html file shows how two model images are used statically.

```
<body>

<div class="north">
     <img src="img/modelibra.jpg" alt="Modelibra" />
</div>

<div class="middle">
     <div wicket:id="aboutMenu">
          To be replaced dynamically by the About menu.
     </div>

     <div class="content">
          <div>
               <img src = "mm/WebLink.jpg" alt="WebLink model" />
          </div>
          <div>
               <img src = "mm/Reference.jpg" alt="Reference model" />
          </div>
     </div>

     <div class="sidebar">
          <div wicket:id="linksCategoryUrlsList">
               To be replaced dynamically by a list of links for the Links
     category.
          </div>
     </div>
</div>

<div class="south">
     Modelibra |
     <a href="http://validator.w3.org/" title="Validate a page">XHTML</a> |
     <a href="http://jigsaw.w3.org/css-validator/" title="Validate CSS">CSS</a> |
     <a href="http://bobby.watchfire.com/">508</a>
</div>

</body>
```

The AboutMenu class inherits its behavior from the DmMenuPanel class of ModelibraWicket. For the FAQ link a FAQ page is created and, in the on click event, the response page is set to the FAQ page. For the Contact Us link the static link is used to postpone the creation of a Contact Us page until a user clicks on the link in the web application.

```
package dmeduc.wicket.app.about;

import org.apache.wicket.markup.html.link.Link;
import org.modelibra.wicket.container.DmMenuPanel;
import org.modelibra.wicket.view.View;
import org.modelibra.wicket.view.ViewModel;

import dmeduc.wicket.weblink.comment.ContactUsPage;
import dmeduc.wicket.weblink.question.FaqPage;

public class AboutMenu extends DmMenuPanel {

	public AboutMenu(final ViewModel viewModel, final View view) {
		super(viewModel, view);
			// FAQ
			Link faqLink = new Link("faqLink") {
				public void onClick() {
					setResponsePage(new FaqPage(viewModel, view));
				}
			};
			add(faqLink);

			// Contact Us
			PageLink contactUsLink = ContactUsPage.link(
				"contactUsLink", viewModel, view);
			add(contactUsLink);
	}

}
```

The FAQ HTML code inherits the HTML code of DmMenuPanel. This is accomplished by the wicket:extend element.

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

<html xmlns:wicket>

<wicket:extend>

	 
	<a class = "button" wicket:id = "faqLink">
		FAQ
	</a>

	 
```

```
      <a class = "button" wicket:id = "contactUsLink">
            <wicket:message key = "contact.us"/>
      </a>

</wicket:extend>

</html>
```

In this way, new HTML elements are added by Wicket to the existing HTML code in the DmMenuPanel.html file, at the place indicated by the wicket:child element. Thus, Wicket supports inheritance of web components both at the Java and the HTML level. In this way, a web component is truly an object oriented component.

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

<html xmlns:wicket>

<wicket:panel>

       
      <a class = "button" wicket:id = "home">
            <wicket:message key = "menu.home"/>
      </a>

       
      <a class = "button" wicket:id = "domain">
            <wicket:message key = "menu.domain"/>
      </a>

       
      <a class = "button" wicket:id = "context">
            <wicket:message key = "menu.context"/>
      </a>

      <wicket:child/>

      <div class="box-title">
            <span wicket:id = "modelName">
                  Model name
            </span>
      </div>

</wicket:panel>

</html>
```

FAQ Page

The FAQ page displays the frequently asked questions about the web application. The page menu is created directly from the DmMenuPanel component of ModelibraWicket. It lets a user return to the

context of the FAQ page, which is the About page. The main content of the page is a list of questions and responses for the FAQ category. The EntityDisplayListPanel component from ModelibraWicket is used to create a simple report. If the FAQ does not exist, there will not be an error. Only a panel will be constructed with the same Wicket id to avoid an error of not having an exact match between a Java component and HTML element for that component. The panel will not be visible and the FAQ will be empty with respect to questions and responses.

```
package dmeduc.wicket.weblink.question;

import org.apache.wicket.markup.html.panel.Panel;
import org.modelibra.wicket.concept.EntityDisplayListPanel;
import org.modelibra.wicket.container.DmMenuPanel;
import org.modelibra.wicket.container.DmPage;
import org.modelibra.wicket.view.View;
import org.modelibra.wicket.view.ViewModel;

import dmeduc.DmEduc;
import dmeduc.weblink.WebLink;
import dmeduc.weblink.category.Categories;
import dmeduc.weblink.category.Category;
import dmeduc.weblink.question.Questions;
import dmeduc.wicket.app.DmEducApp;

public class FaqPage extends DmPage {

	public FaqPage(final ViewModel viewModel, final View view) {
		DmEducApp dmEducApp = (DmEducApp) getApplication();
		DmEduc dmEduc = dmEducApp.getDmEduc();
		WebLink webLink = dmEduc.getWebLink();

		// Menu
		View menuView = new View(dmEducApp);
		menuView.setWicketId("faqMenu");
		menuView.setPage(this);
		menuView.setContextView(view);

		add(new DmMenuPanel(viewModel, menuView));

		// FAQ Category Questions
		ViewModel faqModel = new ViewModel(webLink);
		Categories categories = webLink.getCategories();
		Category faqCategory = categories.getCategoryByName("FAQ");
		if (faqCategory != null) {
			Questions faqCategoryQuestions = faqCategory.getQuestions();
			faqModel.setEntities(faqCategoryQuestions);
		}

		View faqView = new View(dmEducApp);
		faqView.setPage(this);
		faqView.setWicketId("faqCategoryQuestionsList");
```

```
            Panel faq;
            if (faqCategory != null) {
                  faq = new EntityDisplayListPanel(faqModel, faqView);
            } else {
                  faq = new Panel("faqCategoryQuestionsList");
                  faq.setVisible(false);
            }
            add(faq);
      }

}
```

The HTML code for the FAQ page is trivial.

```
<div class="middle">
      <div wicket:id="faqMenu">
            To be replaced dynamically by the FAQ menu.
      </div>

      <div class="content">
            <div wicket:id="faqCategoryQuestionsList">
                  To be replaced dynamically by a list of links for the Links
                  category.
            </div>
      </div>
</div>
```

Contact Us

The Contact Us page displays a form with Text, Source and Creation Date fields. By default the date shows the today's date. The Text field accepts the main message that a user wants to send. The Source field is optional and it may contain any information related to the main message, such as the name of a user, her email or a certain web page.

```
package dmeduc.wicket.weblink.comment;

import org.apache.wicket.Page;
import org.apache.wicket.markup.html.link.IPageLink;
import org.apache.wicket.markup.html.link.PageLink;
import org.apache.wicket.markup.html.panel.FeedbackPanel;
import org.modelibra.wicket.concept.EntityAddFormPanel;
import org.modelibra.wicket.container.DmPage;
import org.modelibra.wicket.security.AccessPoint;
import org.modelibra.wicket.view.View;
import org.modelibra.wicket.view.ViewModel;

import dmeduc.DmEduc;
import dmeduc.weblink.WebLink;
import dmeduc.weblink.comment.Comments;
import dmeduc.weblink.member.Member;
```

```
import dmeduc.wicket.app.DmEducApp;

public class ContactUsPage extends DmPage {

	public ContactUsPage(final ViewModel viewModel, final View view) {
		DmEducApp dmEducApp = (DmEducApp) getApplication();
		DmEduc dmEduc = dmEducApp.getDmEduc();
		WebLink webLink = dmEduc.getWebLink();

		// Menu
		View menuView = new View(dmEducApp);
		menuView.setWicketId("contactUsMenu");
		menuView.setPage(this);
		menuView.setContextView(view);

		add(new DmMenuPanel(viewModel, menuView));

		// Guest user
		if (!getAppSession().isUserSignedIn()) {
			Member guest = new Member(webLink);
			guest.setCode(AccessPoint.GUEST);
			guest.setPassword(AccessPoint.GUEST);
			getAppSession().authenticate(guest, AccessPoint.CODE,
					AccessPoint.PASSWORD);
		}

		// Contact Us
		add(new FeedbackPanel("contactUsFeedback"));

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

		View contactView = new View(dmEducApp);
		contactView.setPage(this);
		contactView.setWicketId("contactUs");
		contactView.setContextView(view);
		add(new EntityAddFormPanel(contactModel, contactView));
	}

	public static PageLink link(final String linkId, final ViewModel viewModel,
			final View view) {
		PageLink link = new PageLink(linkId, new IPageLink() {
			public Page getPage() {
				return new ContactUsPage(viewModel, view);
			}

			public Class<? extends Page> getPageIdentity() {
				return ContactUsPage.class;
			}
		});
		return link;
```

```
	}

}
```

The EntityAddFormPanel component from the org.modelibra.wicket.concept package is used to support the page section where the form is shown.

The corresponding HTML code is trivial.

```
<div class="middle">
	<div wicket:id="contactUsMenu">
		To be replaced dynamically by the Contact Us menu.
	</div>

	<div class="content">
		<div wicket:id = "contactUsFeedback"/>
		<div wicket:id="contactUs">
			To be replaced dynamically by a contact add form panel.
		</div>
	</div>
</div>
```

In ModelibraWicket add, update and remove actions must be done by a member with a predefined security role. Since the Contact Us page is there for anybody who uses this web application, a guest user with the casual role must be created in the Member.xml data file. The sign in concept in the specific domain configuration determines what class must be used to create a user, which is here a member.

```
<member oid="1195574803515">
    <code>guest</code>
    <password>guest</password>
    <lastName>Guest</lastName>
    <firstName>Guest</firstName>
    <email>guest@modelibra.org</email>
    <receiveEmail>false</receiveEmail>
    <role>casual</role>
    <startDate>2007-11-20</startDate>
</member>
```

The casual guest is then authenticated to allow the add action. If this were not done, the Contact Us form will have both Text and Source fields without a possibility to enter a value, and the click on the *Save* button would produce the "Add is not allowed." error. Of course, all of this would not be done for an already signed in member. You can try this by commenting the following code.

```
if (!getAppSession().isUserSignedIn()) {
	Member guest = new Member(webLink);
	guest.setCode(AccessPoint.GUEST);
	guest.setPassword(AccessPoint.GUEST);
	getAppSession().authenticate(guest, AccessPoint.CODE,
		AccessPoint.PASSWORD);
}
```

After a casual user has entered her message, she should not be returned to the Home page as the signed in member. This is accomplished in the EntityAddForm class from the dmeduc.wicket.weblink.comment package, by overriding the protected onSubmit method. In the method, if a user is signed in and is a guest, the user will be automatically signed out at the end of the action provoked by the *Save* button.

```
package dmeduc.wicket.weblink.comment;

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

import dmeduc.weblink.member.Member;
import dmeduc.wicket.home.HomePage;

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

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

	protected void onSubmit() {
		super.onSubmit();
		if (!hasError()) {
			signOutGest();
			setResponsePage(HomePage.class);
		}
	}

	protected void onCancel(final ViewModel viewModel, final View view) {
		super.onCancel(viewModel, view);
		signOutGest();
		setResponsePage(HomePage.class);
	}

	private void signOutGest() {
		if (getAppSession().isUserSignedIn()) {
			Member user = (Member) getAppSession().getSignedInUser();
			if (user.getRole().equals(AccessPoint.CASUAL)) {
				if (user.getCode().equals(AccessPoint.GUEST)
					&& user.getPassword().equals(AccessPoint.GUEST)) {
					getAppSession().signOutUser();
				}
			}
		}
	}

}
```

This class is in the specific package for the Comment concept, but it must carry the generic name

identical to the inherited class from the org.modelibra.wicket.concept package. Every single time when an object of a web component from ModelibraWicket is constructed, ModelibraWicket verifies if the class with the same name exists in the specific package using the standard name for the concept in question. In our example, the package is dmeduc.wicket.weblink.comment. This allows an extension of web components from ModelibraWicket with user oriented features that will be used directly from ModelibraWicket. Realize that the EntityAddFormPanel generic component used in the specific ContactUsPage class will refer to the specific EntityAddForm class with the generic name, and not the class with the same name in ModelibraWicket. This could have been accomplished by creating a new specific web component that would reuse components from ModelibraWicket but with much more code and effort.

Sign Up

With respect to the code, the Sign Up page is similar to the Contact Us page. However, its purpose is quite different. It is there to allow a new user to register as a regular member.

```
package dmeduc.wicket.weblink.member;

import org.apache.wicket.Page;
import org.apache.wicket.markup.html.link.IPageLink;
import org.apache.wicket.markup.html.link.PageLink;
import org.apache.wicket.markup.html.panel.FeedbackPanel;
import org.modelibra.wicket.concept.EntityAddFormPanel;
import org.modelibra.wicket.container.DmPage;
import org.modelibra.wicket.security.AccessPoint;
import org.modelibra.wicket.view.View;
import org.modelibra.wicket.view.ViewModel;

import dmeduc.DmEduc;
import dmeduc.weblink.WebLink;
import dmeduc.weblink.member.Member;
import dmeduc.weblink.member.Members;
import dmeduc.wicket.app.DmEducApp;

public class SignUpPage extends DmPage {

	public SignUpPage(final ViewModel viewModel, final View view) {
		DmEducApp dmEducApp = (DmEducApp) getApplication();
		DmEduc dmEduc = dmEducApp.getDmEduc();
		WebLink webLink = dmEduc.getWebLink();

		// Menu
		View menuView = new View(dmEducApp);
		menuView.setWicketId("signUpMenu");
		menuView.setPage(this);
		menuView.setContextView(view);

		add(new DmMenuPanel(viewModel, menuView));
```

```
            // Guest user
            Member guest = null;
            if (!getAppSession().isUserSignedIn()) {
                  guest = new Member(webLink);
                  guest.setCode(AccessPoint.GUEST);
                  guest.setPassword(AccessPoint.GUEST);
                  getAppSession().authenticate(guest, AccessPoint.CODE,
                        AccessPoint.PASSWORD);
            }

            // Contact us
            add(new FeedbackPanel("signUpFeedback"));

            ViewModel signUpModel = new ViewModel(webLink);
            Members members = webLink.getMembers();
            signUpModel.setEntities(members);

            View signUpView = new View(dmEducApp);
            signUpView.setPage(this);
            signUpView.setWicketId("signUp");
            signUpView.setContextView(view);
            add(new EntityAddFormPanel(signUpModel, signUpView));
      }

      public static PageLink link(final String linkId, final ViewModel viewModel,
                  final View view) {
            PageLink link = new PageLink(linkId, new IPageLink() {
                  public Page getPage() {
                        return new SignUpPage(viewModel, view);
                  }

                  public Class<? extends Page> getPageIdentity() {
                        return SignUpPage.class;
                  }
            });
            return link;
      }

}
```

A member form provides several fields to enter some basic information about a new regular member: code, password, last name, first name, email and whether the new member will receive email. The default start date is set to the current date. The role of the new member is regular and it cannot be changed. The code and password will be used in the sign in process. If all required values are valid they can be saved. The Home page is then displayed and a user may sign in immediately. This is not exactly a standard. Usually, an email about this registration is sent first and a user must validate the registration. This is left for advanced users to do as a useful exercise.

In order to enter new member data, a user becomes temporarily a guest. After the new data are saved, the guest is signed out in the EntityAddForm class in the dmeduc.wicket.weblink.member package.

Specific Component

The ParentChildPropertyDisplayListPanel generic component displays one property from a parent entity and one property from a child entity but in a list of parents with the corresponding child list for each parent. This generic component is not very flexible since it displays only one property from both a parent and a child. Ideally, we would like to have a more generic component or a different generic component that will fulfill the requirement of displaying more properties. However, it is always possible to make our own specific component with that requirement. This is exactly what the MemberInterestsListPanel component is. Figure 10.3. displays this specific web component.

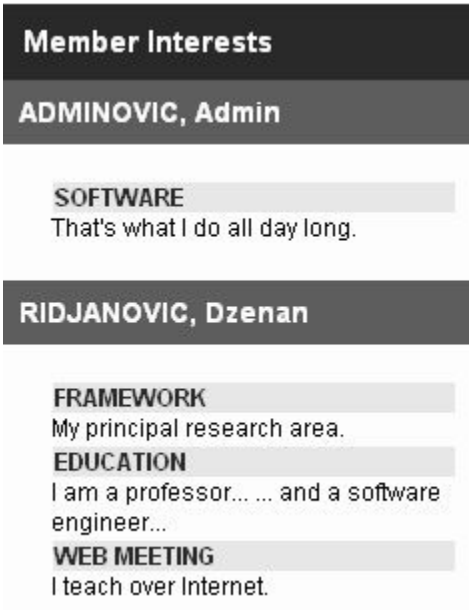

Figure 10.3. Member Interests Section

After the *Member Interests* title, a list of members is displayed. Each member is presented with his last name in capital letters, comma and then his first name. A member has a list of interests. An interest consists of the category name in capital letters and the interest description. In other words, the page sections consists of the section title followed by the list section. The list section is divided in subsections, one for each member. A member subsection consists of the subsection title and a list subsection. The list subsection is further divided into subsections, one for each member interest. An interest subsection contains the category name and the interest description.

The specific web component consists of several Java classes and several HTML files. The name of the component is MemberInterestsListPanel. The name indicates that the component is a panel that will be used for a section of a web page. The member interests are displayed in a list. For each member in the list, there is a list of interests. The first class of the web component is MemberInterestsListPanel.java from the dmeduc.wicket.weblink.member package. This specific web component has two generic arguments in the form of the ViewModel and View classes, Similarly to generic web components of ModelibraWicket. Both arguments are passed to the inheritance parent through the super method.

```
package dmeduc.wicket.weblink.member;

import org.modelibra.wicket.container.DmPanel;
import org.modelibra.wicket.view.View;
```

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

public class MemberInterestsListPanel extends DmPanel {

	public MemberInterestsListPanel(final ViewModel viewModel, final View view) {
		super(viewModel, view);
		ViewModel memberInterestsListModel = new ViewModel();
		memberInterestsListModel.copyPropertiesFrom(viewModel);

		View memberInterestsListView = new View();
		memberInterestsListView.copyPropertiesFrom(view);
		memberInterestsListView.setWicketId("memberInterestsList");

		MemberInterestsList memberList = new MemberInterestsList(
			memberInterestsListModel, memberInterestsListView);
		add(memberList);
	}

}
```

The work done is rather simple. The real work is delegated to the MemberInterestsList class from the same package. Two new arguments are prepared for this new class based on the given ViewModel and View arguments. The only addition is the memberInterestsList Wicket id for the new class. The MemberInterestsListPanel Java class has the corresponding MemberInterestsListPanel HTML file in the same package. The same Wicket id appears in both files.

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

<html xmlns:wicket>

<wicket:panel>

	<div class="section-title">
		Member Interests
	</div>

	<div wicket:id = "memberInterestsList">
		<div wicket:id = "memberName" class = "box-title">
			Member name
		</div>
		<div wicket:id = "interestListPanel">
			List of member interests
		</div>
	</div>

</wicket:panel>

</html>
```

The MemberInterestsListPanel HTML definition can be considered as the section divided into two subsections, one for the specific title, displayed according to the section-title CSS class, and the other

for the section with the Wicket memberInterestsList id.

```
<wicket:panel>

      <div class="section-title">
            Member Interests
      </div>

      <div wicket:id = "memberInterestsList">
            ...
      </div>

</wicket:panel>
```

How is that second section handled is decided in the MemberInterestsList class, which does not have its own HTML code. Its HTML code is given in the parent panel container, which is MemberInterestsListPanel.

```
      <div wicket:id = "memberInterestsList">
            <div wicket:id = "memberName" class = "box-title">
                  Member name
            </div>
            <div wicket:id = "interestListPanel">
                  List of member interests
            </div>
      </div>
```

Thus, we should expect to find two web sub-components in the MemberInterestsList class, one with the memberName Wicket id and the other with the interestListPanel Wicket id. The constructor of the MemberInterestsList class has ViewModel and View arguments. Both arguments are passed to the inheritance parent through the super method. After that, they are memorized as the private properties of the class.

```
package dmeduc.wicket.weblink.member;

import org.apache.wicket.markup.html.basic.Label;
import org.apache.wicket.markup.html.list.ListItem;
import org.modelibra.wicket.container.DmListView;
import org.modelibra.wicket.view.View;
import org.modelibra.wicket.view.ViewModel;

import dmeduc.weblink.interest.Interests;
import dmeduc.weblink.member.Member;
import dmeduc.wicket.weblink.interest.InterestListPanel;


public class MemberInterestsList extends DmListView {

      private ViewModel viewModel;

      private View view;
```

```
	public MemberInterestsList(final ViewModel viewModel, final View view) {
		super(viewModel, view);
		this.viewModel = viewModel;
		this.view = view;
	}

	protected void populateItem(final ListItem item) {
		Member member = (Member) item.getModelObject();
		String lastName = member.getLastName().toUpperCase();
		String firstName = member.getFirstName();
		String memberName = lastName + ", " + firstName;
		Label memberNameLabel = new Label("memberName", memberName);
		item.add(memberNameLabel);

		ViewModel interestModel = new ViewModel();
		interestModel.copyPropertiesFrom(viewModel);
		Interests interests = member.getInterests();
		Interests orderedInterests = interests
			.getInterestsOrderedByCategoryOid(true);
		interestModel.setEntities(orderedInterests);

		View interestView = new View();
		interestView.copyPropertiesFrom(view);
		interestView.setWicketId("interestListPanel");

		InterestListPanel interestListPanel = new InterestListPanel(
			interestModel, interestView);
		item.add(interestListPanel);
	}

}
```

Since the class extends the DmListView class from ModelibraWicket, which extends the ListView class from Wicket, its purpose is to provide a list of objects. This list is handled by the populateItem protected method. The ListView class from Wicket is an abstract class that has the populateItem abstract method. This method, when implemented in a specific class, will be called by Wicket to populate a list of objects. It is the list of entities. More precisely it is the list of Members. How Wicket knows that it is the list of Members?

The DmListView class has the constructor with the ViewModel and View arguments.

```
...
public abstract class DmListView extends ListView {
...
	public DmListView(final ViewModel viewModel, final View view) {
		super(view.getWicketId(), new PropertyModel(viewModel.getEntities(),
				"entityList"));
	}
...
}
```

The Wicket id is obtained from the view and the property model is defined based on entities, which in our case belongs to the Members class, of the view model. The PropertyModel class is from Wicket. It has two arguments. The first argument is an object that has a list property defined by the second argument. In Modelibra, the Entities class has the private property called entityList,

```
private List<T> entityList = Collections.synchronizedList(new ArrayList<T>());
```

which is used by Wicket with the help of Java reflection.

The populateItem method has the ListItem argument. For each item of the list in question, Wicket will call the populateItem method and provide the current item of the list. Thus, the first thing to do is to obtain the current list item in our terms by casting the item to our Member class.

```
Member member = (Member) item.getModelObject();
```

Then, the last name in capital letters is obtained from the member list item, followed by the first name. Both names are concatenated with the comma sign after the last name in capital letters. Wicket has a predefined web component or web widget for displaying a text. It is called Label. The label is constructed with the memberName Wicket id and our grouped member name. Finally, the label is added to the item object.

```
String lastName = member.getLastName().toUpperCase();
String firstName = member.getFirstName();
String memberName = lastName + ", " + firstName;
Label memberNameLabel = new Label("memberName", memberName);
item.add(memberNameLabel);
```

Note that the label is not added to the current object, which is a list, of the MemberInterestsList class, but to the item object, which is a list item. Thus, it is important to use

```
item.add(memberNameLabel);
```

and not

```
add(memberNameLabel);
```

The task of finding a list of interests for a single member is delegated to the InterestListPanel class in the dmeduc.wicket.weblink.interest package. The ViewModel and View arguments are prepared for the class. The member interests are obtained by the getInterests method. The member interests are then ordered by the category oid. It would be more useful to order member interests based on the category name, which is not the property of the Interest concept but the Category parent concept. This task is left to do for advanced readers. The ordered interests are set as the entities of the view model. In addition to the properties copied from the view argument of the MemberInterestsList class, the view has the interestListPanel Wicket id. Finally, the object of the InterestListPanel class is constructed with the prepared arguments, and added to the list item.

```
ViewModel interestModel = new ViewModel();
interestModel.copyPropertiesFrom(viewModel);
```

```
                Interests interests = member.getInterests();
                Interests orderedInterests = interests
                            .getInterestsOrderedByCategoryOid(true);
                interestModel.setEntities(orderedInterests);

                View interestView = new View();
                interestView.copyPropertiesFrom(view);
                interestView.setWicketId("interestListPanel");

                InterestListPanel interestListPanel = new InterestListPanel(
                            interestModel, interestView);
                item.add(interestListPanel);
```

It is important to repeat that it is Wicket that will call the populateItem method as many times as there are members, and it is Wicket that will prepare the member list with the memberInterestsList Wicket id in the MemberInterestsListPanel.html.

```
<div wicket:id = "memberInterestsList">
      <div wicket:id = "memberName" class = "box-title">
            Member name
      </div>
      <div wicket:id = "interestListPanel">
            List of member interests
      </div>
</div>
```

The sub-section with the interestListPanel Wicket id is handled in the InterestListPanel class, which is similar to the MemberInterestsListPanel class. The InterestListPanel class also extends the DmPanel class and has the ViewModel and View arguments.

```
package dmeduc.wicket.weblink.interest;

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

public class InterestListPanel extends DmPanel {

      public InterestListPanel(final ViewModel viewModel, final View view) {
            super(viewModel, view);
            ViewModel interestListModel = new ViewModel();
            interestListModel.copyPropertiesFrom(viewModel);

            View interestListView = new View();
            interestListView.copyPropertiesFrom(view);
            interestListView.setWicketId("interestList");

            InterestList interestList = new InterestList(interestListModel,
                  interestListView);
            add(interestList);
      }
```

```
}
```

The InterestListPanel class delegates the task of finding a list of interests for a single member to the InterestList class, which has the interestList Wicket id. The corresponding InterestListPanel HTML file uses that Wicket id to connect the HTML section with the Java code.

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

<html xmlns:wicket>

<wicket:panel>

     <ul>
          <li wicket:id = "interestList">
               <div wicket:id = "categoryName" class = "marker">
                    Category name
               </div>
               <div wicket:id = "interestDescription">
                    Interest description
               </div>
          </li>
     </ul>

</wicket:panel>

</html>
```

The list of interests for a single member is prepared by the populateItem method of the InterestList class. The item method argument is casted to the interest object of the Interest class. From the interest object the category parent is obtained by the getCategory method. The category name is then transformed to the upper case, and is used as the categoryName label for the interest. The label is then added to the item object.

The interest entity and the description property are set in the view model for the interestDescription view for the LabelPanel text component from the org.modelibra.wicket.text package. The object of the LabelPanel class is constructed by using the two standard arguments and it is added to the item object.

```
package dmeduc.wicket.weblink.interest;

import org.apache.wicket.markup.html.basic.Label;
import org.apache.wicket.markup.html.list.ListItem;
import org.modelibra.wicket.container.DmListView;
import org.modelibra.wicket.text.LabelPanel;
import org.modelibra.wicket.view.View;
import org.modelibra.wicket.view.ViewModel;

import dmeduc.weblink.category.Category;
import dmeduc.weblink.interest.Interest;

public class InterestList extends DmListView {
```

```
	private ViewModel viewModel;

	private View view;

	public InterestList(final ViewModel modelContext, final View viewContext) {
		super(modelContext, viewContext);
		this.viewModel = modelContext;
		this.view = viewContext;
	}

	protected void populateItem(final ListItem item) {
		Interest interest = (Interest) item.getModelObject();

		Category category = interest.getCategory();
		String title = category.getName().toUpperCase();
		Label categoryNameLabel = new Label("categoryName", title);
		item.add(categoryNameLabel);

		ViewModel interestModel = new ViewModel();
		interestModel.copyPropertiesFrom(viewModel);
		interestModel.setEntity(interest);
		interestModel.setPropertyCode("description");

		View interestView = new View();
		interestView.copyPropertiesFrom(view);
		interestView.setWicketId("interestDescription");

		item.add(new LabelPanel(interestModel, interestView));
	}

}
```

This time the LabelPanel component from ModelibraWicket is used instead of the Label component from Wicket, just to show you that for every basic web component (or web widget) in Wicket there is a corresponding basic web component in ModelibraWicket. The basic component in ModelibraWicket is a panel and it has the ViewModel and View arguments to make a strong connection between the web component and Modelibra for a view model and ModelibraWicket for a view. These types of basic components, which are all panels with the predefined two arguments, are used in ModelibraWicket to develop more advanced web components in a generic way.

Summary

The web application has several new pages that are considered standard: *About*, *FAQ*, *Contact Us* and *Sign Up*. The *About* page displays two domain models and provides a list of web links for the Links category. From the *About* page a user may go to the *FAQ* or *Contact Us* page. The *FAQ* page lists questions and responses for the FAQ category. The *Contact Us* page provides a form for a user comment. The *Sign Up* page allows a new user to register as a regular member.

The home page of the web application has three new web sections. The *Sign In* section is displayed at

the beginning of the right column of the home page, The *I18n* section is a drop down choice that is a part of the menu of the home page. A user may change the language from English to French or Bosnian. English is the default language. The *Countries* section is located at the end of the right column of the home page. The section component is a table with country codes and names. The component provides the block bar so that only a small number of countries may be seen at once.

The specific web component, which displays a list of member names, is extended with member interests.

There is a new model in the domain. The model is named Reference because it is used for reference purposes. It has three concepts: CountryName, CountryLanguage and SecurityRole. Only the last two concepts are essential for the application. The country languages are used in the I18n component, and the security roles are used to determine the rights of the signed in member.

The security roles determine if users have display and update rights for concepts, properties and neighbors. The display of concepts, properties and neighbors may be done without sign in. Without sign in, all users have the same display rights determined by the display configurations of concepts, properties and neighbors. With sign in, display and update rights are determined by the security roles. The security roles are: admin, manager, advanced, regular and casual. The admin role does not have any restrictions, while the casual role has many.

The next chapter will focus on specific web components developed in ModelibraWicket and Wicket.

Questions

1. What is the purpose of the Reference model?

2. Can you override display and update rights in the specific configuration?

3. What must be done to include the support for a new natural language?

4. How can you breach the security of the *Sign Up* page?

Exercises

Exercise 10.1.

Create the FAQ page without using any ModelibraWicket components. In other words use Modelibra for the model and Wicket only for the view.

Exercise 10.2.

Create a page for the Reference model so that security roles are displayed together with your explanation of each role.

Exercise 10.3.

Create a page so that a hierarchy of all categories is displayed together with web links for each category.

Web Links

[Wicket JavaDoc] Wicket JavaDoc
http://people.apache.org/~tobrien/wicket/apidocs/index.html