Chapter 11: Specific Web Components

The objective of this chapter is to provide more user oriented web pages and web components. The corresponding spiral is DmEduc-09. After a member has signed in, he may enter his page by clicking on the *My Page* link. The page shows his interests. For each interest there is a *Category* link to show the approved category with its approved web links. The member may edit his interests and maintain the corresponding web links. There is a new link on the home page called *Question Selection*. A subset of questions may be selected based on keywords that appear in the *Text* field. The *Sign Up* page enables a new user to register. This time he will receive an email with the confirmation number that he will have to enter in order to become a regular member. This is an improvement with respect to the previous spiral, where a casual member was admitted without an email validation. Finally, a tree of categories may be expanded on the home page to see web links of a certain subcategory, no matter how deep that subcategory is in the hierarchy of categories. In addition a new subcategory may be added to a parent category somewhere in the tree.

Domain Model

The Web Link model from the previous spiral has not been changed in ModelibraModeler (Figure 11.1). However, there is a new concept in the XML specific configuration of the Web Link model. The concept is named Applicant and is used to temporarily save a new member until the email confirmation is approved. This is not the best way to handle changes in the model, but changes like that happen quite often in the real world and it is important to show an example of that type of change.

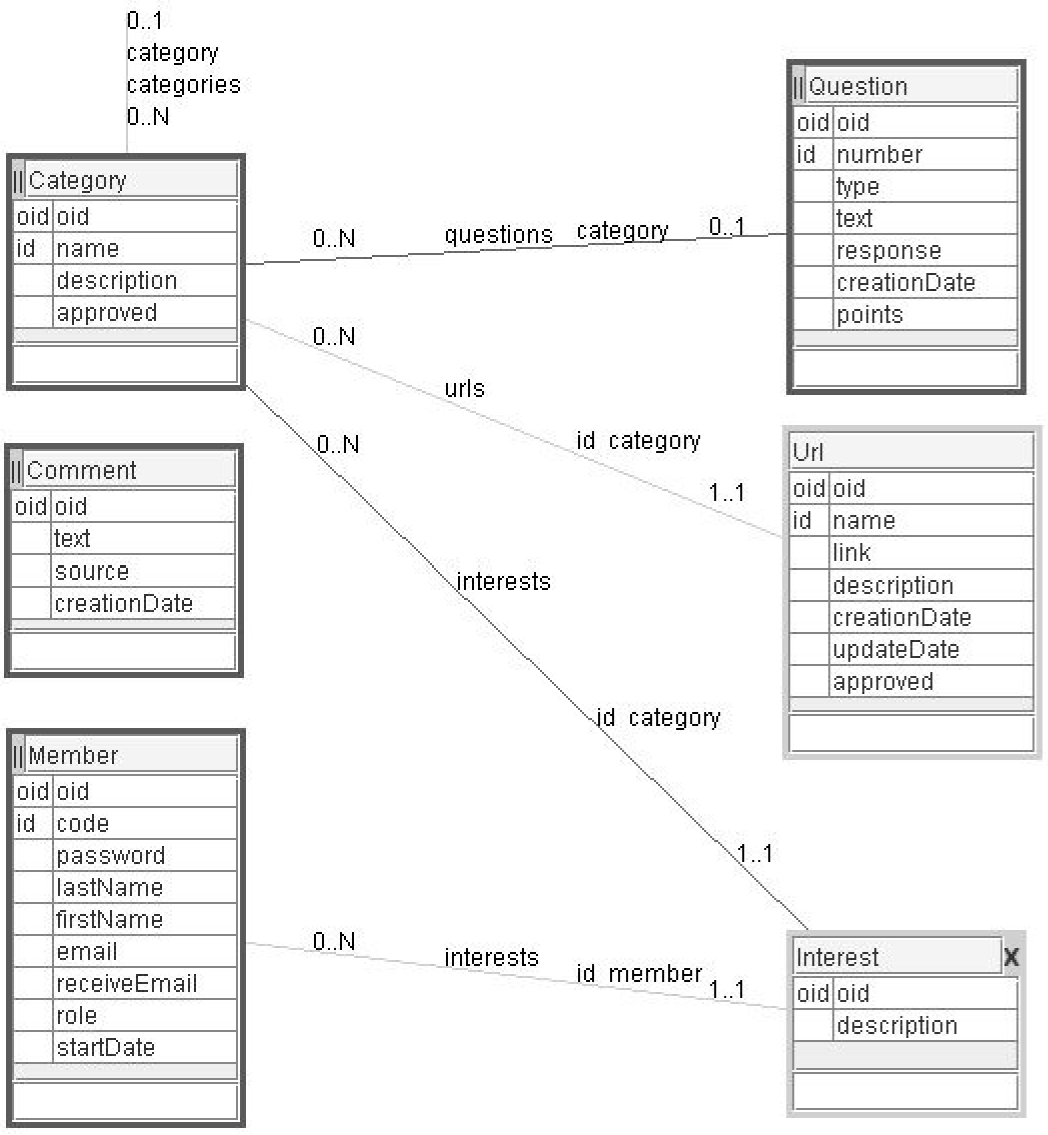

Figure 11.1. Web Link Model

Domain Configuration

The Applicant concept extends the Member concept. Since the Member concept exists in the specific configuration, the Applicant concept actually inherits properties of the Member concept in the specific configuration, which in turn extends the Member concept in the reusable configuration.

```
<concept oid="1172170727630">
<code>Applicant</code>
<extension>true</extension>
<extensionConcept>Member</extensionConcept>
<entitiesCode>Applicants</entitiesCode>

<add>true</add>
<remove>true</remove>
```

```
			<update>false</update>
		</concept>
```

Realize that an applicant can be added and removed but not updated.

Code Generation

There is no code generated in this spiral. The code generation is not used to show that for certain things that will not likely change in future it may be easier to open the specific code. Thus, all code necessary to support the Applicant concept is done by hand. For example, the specific code is added to the WebLink class.

```
package dmeduc.weblink;

import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;
import org.modelibra.IDomain;

import dmeduc.weblink.applicant.Applicants;
import dmeduc.weblink.category.Categories;
import dmeduc.weblink.category.Category;
import dmeduc.weblink.url.Url;
import dmeduc.weblink.url.Urls;

public class WebLink extends GenWebLink {

	private static Log log = LogFactory.getLog(WebLink.class);

	/* ============================ */
	/* ======= SPECIFIC CODE ======= */
	/* ============================ */

	private Applicants applicants;

	public WebLink(IDomain domain) {
		super(domain);
		applicants = new Applicants(this);
	}

	public Applicants getApplicants() {
		return applicants;
	}

	...

}
```

There is a new package with only Applicant and Applicants classes. The Applicant class extends the specific Member class.

```java
package dmeduc.weblink.applicant;

import org.modelibra.IDomainModel;

import dmeduc.weblink.member.Member;

public class Applicant extends Member {

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

}
```

The Applicants class extends the specific Members class. The preAdd specific method does all necessary validations for a new member by calling the preAdd method of its inheritance parent. If everything is correct then the additional validation is done to be sure that the applicant code does not already exist among current members.

```java
package dmeduc.weblink.applicant;

import org.modelibra.IDomainModel;

import dmeduc.weblink.WebLink;
import dmeduc.weblink.member.Member;
import dmeduc.weblink.member.Members;

public class Applicants extends Members {

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


	protected boolean preAdd(Member member) {
		if (super.preAdd(member)) {
			Applicant applicant = (Applicant) member;
			WebLink webLink = (WebLink) getModel();
			Members members = webLink.getMembers();
			Member existingMember =
				members.getMemberByCode(applicant.getCode());
			if (existingMember == null) {
				return true;
			} else {
				getErrors()
					.add("Applicant.id.unique", "Member code already exists.");
				return false;
			}
		} else {
			return false;
```

```
            }
        }

}
```

The specific property keys for the Applicant concept are added manually in the DmEduc_en.properties file (and other i18n files). The corresponding messages from the Member concept are just copied.

```
Applicant=Member
Applicants=Members
Applicant.id=Member identifier: ([code] [])
Applicant.id.unique=Member code is not unique.
Applicant.code=Code
Applicant.code.required=Code is required.
Applicant.code.length=Code is longer than 16.
Applicant.password=Password
Applicant.password.required=Password is required.
Applicant.password.length=Password is longer than 16.
Applicant.lastName=LastName
Applicant.lastName.required=LastName is required.
Applicant.lastName.length=LastName is longer than 32.
Applicant.firstName=FirstName
Applicant.firstName.required=FirstName is required.
Applicant.firstName.length=FirstName is longer than 32.
Applicant.email=Email
Applicant.email.required=Email is required.
Applicant.email.length=Email is longer than 80.
Applicant.email.validation=Email is not a valid org.modelibra.type.Email value.
Applicant.receiveEmail=ReceiveEmail
Applicant.receiveEmail.required=ReceiveEmail is required.
Applicant.role=Role
Applicant.role.required=Role is required.
Applicant.role.length=Role is longer than 16.
Applicant.startDate=StartDate
Applicant.startDate.required=StartDate is required.
Applicant.startDate.length=StartDate is longer than 16.
```

Sign Up

The Sign Up page in this spiral uses the Applicant entities in the ViewModel argument of the generic EntityAddFormPanel web component. Note that the SignUpPage class moved to the new dmeduc.wicket.weblink.applicant package.

```
package dmeduc.wicket.weblink.applicant;

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.applicant.Applicant;
import dmeduc.weblink.applicant.Applicants;
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();

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

            // Sign up
            add(new FeedbackPanel("signUpFeedback"));

            ViewModel signUpModel = new ViewModel(webLink);
            Applicants applicants = webLink.getApplicants();
            signUpModel.setEntities(applicants);

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

      ...

}
```

The Applicant class is used for the guest user, so that he can be authenticated by ModelibraWicket. The authentication is done by comparing the code and password of the guest user with the casual member of the Member entry point of the WebLink model.

The EntityAddFormPanel component is generic but it verifies if there is a specific subcomponent, for the Applicant concept, which carries the generic name of EntityAddForm. This specific component, if it exists, must be located in the dmeduc.wicket.weblink.applicant package. The specific component is created to add some specific code to the generic component. The form has two buttons, one to add an applicant and the other to cancel the operation. Since the guest member is used to add the applicant, the cancel operation signs out the guest user.

```
package dmeduc.wicket.weblink.applicant;

import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;
import org.modelibra.config.DomainConfig;
import org.modelibra.util.EmailConfig;
import org.modelibra.wicket.security.AccessPoint;
import org.modelibra.wicket.util.LocalizedText;
import org.modelibra.wicket.view.View;
import org.modelibra.wicket.view.ViewModel;

import dmeduc.weblink.WebLink;
import dmeduc.weblink.applicant.Applicant;
import dmeduc.weblink.applicant.Applicants;
import dmeduc.weblink.member.Member;
import dmeduc.wicket.app.DmEducApp;
import dmeduc.wicket.app.home.HomePage;

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

	private static Log log = LogFactory.getLog(EntityAddForm.class);

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

	protected void onSubmit(final ViewModel viewModel, final View view) {
		super.onSubmit(viewModel, view);
		Applicant applicant = (Applicant) viewModel.getEntity();
		WebLink webLink = (WebLink) viewModel.getModel();
		Applicants applicants = webLink.getApplicants();
		if (applicants.contain(applicant)) {
			sendEmailToConfirm(viewModel);
			setResponsePage(ConfirmationPage.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();
				}
			}
		}
	}

	private void sendEmailToConfirm(final ViewModel viewModel) {
		DmEducApp app = (DmEducApp) getApplication();
		DomainConfig domainConfig = (DomainConfig) app.getDomain()
				.getDomainConfig();
		EmailConfig emailConfig = domainConfig.getConfig().getEmailConfig();

		String messageSubject = LocalizedText.getText(this,
				"signUp.message.subject");
		String messageStart = LocalizedText.getText(this,
				"signUp.message.start");
		Applicant applicant = (Applicant) viewModel.getEntity();
		Long confirmationNumber = applicant.getOid().getUniqueNumber();
		String messageFinish = LocalizedText.getText(this,
				"signUp.message.finish");

		applicant.emailMessage(emailConfig, messageSubject, messageStart + " "
				+ confirmationNumber + messageFinish);
	}

}
```

Note that the Member class is used in the signOutGest method since the signin concept in the specific domain configuration is Member. When an applicant is signed in as a casual user, the member entity with the same code and password as the applicant will be found and referenced as the signed in user. Thus, if the Applicant class had been used instead of the Member class in the signOutGest method, the casting exception would have been raised.

```
	<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 onSubmit method first calls the same method of its inheritance parent that hopefully adds a new applicant. The specific method then verifies if the applicants object contains the applicant in order to send him an email to confirm the registration. The private sendEmailToConfirm method obtains the email configuration from the domain configuration and prepares the message text for the emailMessage method inherited from the Member class.

```
	public void emailMessage(EmailConfig emailConfig, String subject,
			String message) {
		try {
			emailConfig.send(getEmail().toString(), subject, message);
		} catch (ModelibraException e) {
			log.error("Error in Member.emailMessage: " + e.getMessage());
		}
	}
```

The email configuration can be found in the config subdirectory of the WEB-INF directory. Of course, the dddddddd value is not a real password.

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

<emails>
	<email oid="1">
		<code>vlgiiora</code>
		<toSendEmail>yes</toSendEmail>
		<from>dzenan.ridjanovic@videotron.ca</from>
		<outServer>relais.videotron.ca</outServer>
		<password>dddddddd</password>
	</email>
</emails>
```

A new user must enter the confirmation number in the corresponding field within the confirmation page.

```
package dmeduc.wicket.weblink.applicant;

import org.modelibra.wicket.container.DmPage;
import org.modelibra.wicket.util.LocalizedText;

public class ConfirmationPage extends DmPage {

	public ConfirmationPage() {
		ConfirmationPanel confirmationPanel = new ConfirmationPanel(
				"confirmation");
		add(confirmationPanel);
		String invalidMessage = LocalizedText.getText(confirmationPanel,
				"invalid");
		confirmationPanel.setInvalidMessage(invalidMessage);
	}

}
```

The ConfirmationPage class from the dmeduc.wicket.weblink.applicant package has only a confirmation panel. After the confirmation panel is constructed its invalid message is set in order to avoid a warning from Wicket that a localized message should not be searched within a constructor of the component where the message will be displayed.

The ConfirmationPanel specific web component creates a confirmation form within its constructor. The unconfirmLabel property is used to accept the invalid message from its context. The private ConfirmationForm class is so called inner class [Inner Class]. An inner class is nested within its container class as a support class needed only locally. Here, the ConfirmationForm class that extends the Form class from Wicket is created to display a single text field where a user will enter a confirmation number.

```
package dmeduc.wicket.weblink.applicant;

import org.apache.wicket.markup.html.basic.Label;
import org.apache.wicket.markup.html.form.Form;
import org.apache.wicket.markup.html.form.TextField;
import org.apache.wicket.model.Model;
import org.modelibra.util.Transformer;
import org.modelibra.wicket.container.DmPanel;
import org.modelibra.wicket.security.AccessPoint;

import dmeduc.DmEduc;
import dmeduc.weblink.WebLink;
import dmeduc.weblink.applicant.Applicant;
import dmeduc.weblink.applicant.Applicants;
import dmeduc.weblink.member.Member;
import dmeduc.weblink.member.Members;
import dmeduc.wicket.app.DmEducApp;
import dmeduc.wicket.app.home.HomePage;

public class ConfirmationPanel extends DmPanel {

      private Label unconfirmLabel;

      public ConfirmationPanel(final String wicketId) {
            super(wicketId);

            ConfirmationForm confirmationForm = new ConfirmationForm(
                        "confirmationForm");
            add(confirmationForm);

            unconfirmLabel = new Label("unconfirm", "");
            unconfirmLabel.setVisible(false);
            add(unconfirmLabel);
      }

      public void setInvalidMessage(String message) {
            unconfirmLabel.setModelObject(message);
      }

      private class ConfirmationForm extends Form {
            private String oidString = "";

            private TextField oidField;

            public ConfirmationForm(final String wicketId) {
                super(wicketId);

                oidField = new TextField("oidField", new Model(oidString));
                add(oidField);
            }

            protected void onSubmit() {
                Applicant applicant;
                oidString = oidField.getModelObjectAsString();
                Long oid = Transformer.longInteger(oidString);
                if (oid == null) {
                    unconfirmLabel.setVisible(true);
                 } else {
                    DmEducApp dmEducApp = (DmEducApp) getApplication();
                    DmEduc dmEduc = dmEducApp.getDmEduc();
                    WebLink webLink = dmEduc.getWebLink();

                    Applicants applicants = webLink.getApplicants();
                    applicant = (Applicant) applicants.getMember(oid);

                    if (applicant != null) {
                             unconfirmLabel.setVisible(false);
                             Members members = webLink.getMembers();
                             Member newMember = new Member(members.getModel());
                             newMember.copyFromApplicant(applicant);
                             members.add(newMember);
                             applicants.remove(applicant);
                             signOutGest();
                             setResponsePage(HomePage.class);
                       } else {
                             unconfirmLabel.setVisible(true);
                       }
                 }
            }

      }

      ...

  }
```

The constructor of the TextField widget used from Wicket has two arguments. The first argument is a Wicket identifier that is used to connect a Java component with an HTML element. The following is the content of the ConfirmationPanel.html file.

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

```
<html xmlns:wicket>

<wicket:panel>

      <form wicket:id = "confirmationForm">
            <table>
                  <caption>
                        <wicket:message key = "signUp.confirmation.title"/>
                  </caption>
                  <tr>
                        <th align = "right">
                              <wicket:message key = "signUp.confirmation.oid"/>
                        </th>
                        <td>
                              <input wicket:id = "oidField"
                                          type = "text" value = "?"
                                          size = "20" />
                        </td>emailConfig.send(getEmail().toString(), subject, message);
                  </tr>

                  <tr>
                        <td>

                        </td>
                        <td>
                              <wicket:message key = "signUp.confirmation.confirm"/>
                              <input type = "submit" value = "+"/>
                        </td>
                  </tr>
            </table>
      </form>

      <br/>
      <div wicket:id = "unconfirm" class="marker">
            Non-confirmation message
      </div>
      <br/>

</wicket:panel>

</html>
```

The second argument of the TextField widget is of the Model class from Wicket. The model for a text field is a String object, here the oidString property.

The onSubmit method fetches first a confirmation number provided by a user as an ordinary text. This is done by using the getModelObjectAsString method of the TextField class. If this text is not a Long integer, the invalid message from the ConfirmationPanel.properties file will be displayed:

```
invalid=The value is not valid.
```

```
signUp.confirmation.confirm=Confirm
signUp.confirmation.oid=Confirmation Number
signUp.confirmation.title=Confirmation
```

If the provided value is valid it is used as an oid to retrieve an applicant so that he can be moved from applicants to members. At the end, the guest user, which allows both add and remove operations, is signed out.

My Page

After a member has signed in, he may enter his page by clicking on the *My Page* link. The page is divided into two sections: the parent section displays the member's name and the child section shows his interests for categories of web links. Thus, the specific MemberPage class is of the parent-child type. The specific page is composed of two generic web components from ModelibraWicket: EntityDisplayMinPanel and EntityUpdateTablePanel.

```
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.modelibra.wicket.concept.EntityDisplayMinPanel;
import org.modelibra.wicket.container.DmUpdatePage;
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.EntityUpdateTablePanel;

public class MemberPage extends DmUpdatePage {

	public MemberPage(final ViewModel viewModel, final View view) {
		super(viewModel, view);
		ViewModel memberPageModel = new ViewModel();
		memberPageModel.copyPropertiesFrom(viewModel);
		View memberPageView = new View();
		memberPageView.copyPropertiesFrom(view);
		memberPageView.setContextView(view);
		memberPageView.setPage(this);

		// Member
		ViewModel memberModel = new ViewModel();
		memberModel.copyPropertiesFrom(memberPageModel);
		View memberView = new View();
		memberView.copyPropertiesFrom(memberPageView);
		memberView.setWicketId("minMemberSection");
		add(new EntityDisplayMinPanel(memberModel, memberView));
```

```
        // Member Interests
        ViewModel membernterestsModel = new ViewModel();
        membernterestsModel.copyPropertiesFrom(memberPageModel);
        Member member = (Member) viewModel.getEntity();
        Interests memberInterests = member.getInterests();
        membernterestsModel.setEntities(memberInterests);
        membernterestsModel.setEntity(null);
        View memberInterestsView = new View();
        memberInterestsView.copyPropertiesFrom(memberPageView);
        memberInterestsView.setContextView(memberPageView);
        memberInterestsView.setUpdate(true);
        memberInterestsView.setWicketId("memberInterestsSection");
        add(new EntityUpdateTablePanel(membernterestsModel,
                memberInterestsView));
    }

    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 MemberPage(viewModel, view);
            }

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

}
```

The MemberPage class extends the DmUpdatePage class from ModelibraWicket. As a consequence, a user must sign in before accessing the page where he may edit his interests and maintain the corresponding web links. The first thing done in the constructor of the MemberPage class is a preparation of the page context that will be used to prepare ViewModel and View arguments of the two generic web components used for parent and child sections of the page. The context for the EntityDisplayMinPanel component is trivial. A user is encouraged to experiment with additional generic web components from ModelibraWicket such as EntityDisplayPanel and EntityUpdateFormPanel.

The context for the EntityUpdateTablePanel component is a bit more elaborate. A member is found in the viewModel argument of the page and his interests are obtained and set as entities of the component's model context. The view context of the component borrows its definition from the view context of the page.

It should be obvious that even for a page of the parent-child type, when generic web components from ModelibraWicket are used, the Java code is minimal and not complex. Similarly the HTML code is simple.

```
<body>

      <wicket:extend>

      <div class="middle">
            <div class="content">
                  <br/>
                  <div wicket:id="minMemberSection">
                        To be replaced dynamically by min. info about the member.
                  </div>
                  <br/>
                  <div wicket:id="memberInterestsSection">
                        To be replaced dynamically by the member interests.
                  </div>
                  <br/>
            </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>

      </wicket:extend>

</body>
```

The following is the code used in the MainMenu class to provide a link to *My Page*.

```
                  // Member Page
                  ViewModel myViewModel = new ViewModel(webLink);
                  myViewModel.setEntities(members);
                  View myView = new View();
                  myView.setContextView(view);
                  myView.setPage(view.getPage());
                  Link myLink = MemberPage.link("myLink", myViewModel, myView);
                  add(myLink);
                  if (getAppSession().isUserSignedIn()) {
                        Member signedInMember = (Member) getAppSession()
                              .getSignedInUser();
                        myViewModel.setEntity(signedInMember);
                  } else {
                        myLink.setVisible(false);
                  }
```

Category Urls

For each member interest in the MemberPage class there is a *Category* link to show the approved category with its approved web links. For the Interest concept the Category concept is one of its two parents. Since in generic web components of ModelibraWicket only links to child concepts are provided in navigations from concept to concept, the Category link must be added as a specific code. This is done in the EntityUpdateTableListView view component for the Interests concept.

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

import org.apache.wicket.markup.html.list.ListItem;
import org.modelibra.wicket.view.View;
import org.modelibra.wicket.view.ViewModel;

import dmeduc.weblink.WebLink;
import dmeduc.weblink.category.Categories;
import dmeduc.weblink.category.Category;
import dmeduc.weblink.interest.Interest;
import dmeduc.wicket.weblink.category.CategoryUrlsPage;

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

	private ViewModel viewModel;

	private View view;

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

	protected void populateItem(final ListItem item) {
		super.populateItem(item);
		Interest memberInterest = (Interest) item.getModelObject();
		Category memberInterestCategory = memberInterest.getCategory();
		ViewModel categoryPageModel = new ViewModel();
		WebLink webLink = (WebLink) viewModel.getModel();
		Categories categories = webLink.getCategories();
		categoryPageModel.copyPropertiesFrom(viewModel);
		categoryPageModel.setEntities(categories);
		categoryPageModel.setEntity(memberInterestCategory);
		item.add(CategoryUrlsPage.link("categoryUrlsLink", categoryPageModel,
			view));
	}

}
```

The protected populateItem method is overridden. The method calls first its counterpart in the inheritance parent that populates a row in a table of interests up to a specific link. In the specific code, the item object is casted to an interest and the interest's category is retrieved. The categories entry point is obtained from its model. Both category and categories are set in a view model for the

CategoryUrlsPage specific class.

The CategoryUrlsPage class is similar to the MemberPage class. It is also of the parent-child type. The parent is a single category that is displayed by the EntityDisplayPanel generic web component. The child is a table of urls that may be updated. The table is provided by the EntityUpdateTablePanel generic web component.

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

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.Panel;
import org.modelibra.wicket.concept.EntityDisplayPanel;
import org.modelibra.wicket.concept.EntityUpdateTablePanel;
import org.modelibra.wicket.container.DmUpdatePage;
import org.modelibra.wicket.view.View;
import org.modelibra.wicket.view.ViewModel;

import dmeduc.weblink.category.Category;
import dmeduc.weblink.url.Urls;

public class CategoryUrlsPage extends DmUpdatePage {

	public CategoryUrlsPage(final ViewModel viewModel, final View view) {
		super(viewModel, view);
		ViewModel categoryUrlsPageModel = new ViewModel();
		categoryUrlsPageModel.copyPropertiesFrom(viewModel);
		View categoryUrlsPageView = new View();
		categoryUrlsPageView.copyPropertiesFrom(view);
		categoryUrlsPageView.setContextView(view);
		categoryUrlsPageView.setPage(this);

		// Category
		ViewModel categoryModel = new ViewModel();
		categoryModel.copyPropertiesFrom(categoryUrlsPageModel);
		View categoryView = new View();
		categoryView.copyPropertiesFrom(categoryUrlsPageView);
		categoryView.setWicketId("categorySection");
		add(new EntityDisplayPanel(categoryModel, categoryView));

		// Category Urls
		ViewModel categoryUrlsModel = new ViewModel();
		categoryUrlsModel.copyPropertiesFrom(categoryUrlsPageModel);
		Category category = (Category) viewModel.getEntity();
		Urls categoryUrls = category.getUrls().getApprovedUrls();
		categoryUrlsModel.setEntities(categoryUrls);
		categoryUrlsModel.setEntity(null);
		View categoryUrlsView = new View();
		categoryUrlsView.copyPropertiesFrom(categoryUrlsPageView);
		categoryUrlsView.setContextView(categoryUrlsPageView);
```

```
        categoryUrlsView.setUpdate(true);
        categoryUrlsView.setWicketId("categoryUrlsSection");
        Panel categoryUrlsPanel;
        if (category.isApproved()) {
            categoryUrlsPanel = new EntityUpdateTablePanel(
                categoryUrlsModel, categoryUrlsView);
        } else {
            categoryUrlsPanel = new Panel("categoryUrlsSection");
            categoryUrlsPanel.setVisible(false);
        }
        add(categoryUrlsPanel);
    }

    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 CategoryUrlsPage(viewModel, view);
            }

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

}
```

The first thing done in the constructor of the CategoryUrlsPage class is a preparation of the page context that will be used to prepare ViewModel and View arguments of the two generic web components used for parent and child sections of the page. The context for the EntityDisplayPanel component is trivial.

The context for the EntityUpdateTablePanel component is a bit more elaborate. A category is found in the viewModel argument of the page and its approved urls are obtained and set as entities of the component's model context. The view context of the component borrows its definition from the view context of the page. Only the approved urls are presented, but only if the category is approved. If a category is not approved a page section of the Panel class from Wicket is created with the same Wicket identifier, categoryUrlsSection, and then set to invisible.

Question Selection

In the HomeMenu class there is a link to the EntityPropertyKeywordSelectPage generic web page from the org.modelibra.wicket.concept.selection package. This is an example of how a page may be reused as a web component. The web page is informed about the questions entry point into the model and about the text property of the Question concept that is used to select questions based on a single keyword used somewhere in the text of the text property, or based on several keywords separated by the comma sign used in a some or all expression.

```
// Select Questions
ViewModel selectQuestionsViewModel = new ViewModel(webLink);
Questions questions = webLink.getQuestions();
selectQuestionsViewModel.setEntities(questions);
selectQuestionsViewModel.setPropertyCode("text");
View selectQuestionsView = new View();
selectQuestionsView.setContextView(view);
selectQuestionsView.setPage(view.getPage());
final WebPage selectQuestionsPage = new
      EntityPropertyKeywordSelectPage(
            selectQuestionsViewModel, selectQuestionsView);
Link selectQuestionsLink = new Link("selectQuestionsLink") {
      public void onClick() {
            setResponsePage(selectQuestionsPage);
      }
};
add(selectQuestionsLink);
```

Category Tree

The HomePage class has a tree section where a hierarchy of categories can be expanded to see categories and subcategories each with its own web links, with a possibility to display the specific Category Urls page or to even add a new subcategory.

```
// Category Tree
ViewModel treeViewModel = new ViewModel(webLink);
Categories categories = webLink.getCategories();
treeViewModel.setEntities(categories);
View treeView = new View();
treeView.setPage(this);
treeView.setContextView(homePageView);
treeView.setWicketId("categoryTree");
add(new CategoryTreePanel(treeViewModel, treeView));
```

The invocation context of the CategoryTreePanel web component is trivial. Of course, the categories object must be part of a reflexive concept. As seen by its name, the web component is specific. This is an example of how a specific web component may be created based on Modelibra and Wicket only.

The CategoryTreePanel class extends the Panel component from Wicket. It has the ViewModel and View arguments from ModelibraWicket, but the component could have been made with specific arguments only. The Wicket identifier from the view argument is passed to the constructor of the Panel class. A Java list of approved categories is passed as a part of the second argument of the LinkTree component from Wicket.

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

import java.util.List;
import javax.swing.tree.DefaultMutableTreeNode;
import javax.swing.tree.DefaultTreeModel;

import org.apache.wicket.Component;
import org.apache.wicket.markup.html.panel.Panel;
import org.apache.wicket.markup.html.tree.LinkTree;
import org.apache.wicket.model.IModel;
import org.modelibra.wicket.view.View;
import org.modelibra.wicket.view.ViewModel;

import dmeduc.weblink.category.Categories;
import dmeduc.weblink.category.Category;

public class CategoryTreePanel extends Panel {

    public CategoryTreePanel(final ViewModel viewModel, final View view) {
        super(view.getWicketId());
        final Categories categories = (Categories) viewModel.getEntities();
        List<Category> categoryList = categories.getApprovedCategories()
            .getList();

        LinkTree tree = new LinkTree("tree", convertToTreeModel(categoryList)) {
            @Override
            protected Component newNodeComponent(final String id,
                    final IModel model) {
                View nodeView = new View();
                nodeView.setPage(getPage());
                nodeView.setWicketId(id);
                nodeView.setContextView(nodeView);

                ViewModel nodeViewModel = new ViewModel();
                nodeViewModel.copyPropertiesFrom(viewModel);
                nodeViewModel.setEntities(categories);

                DefaultMutableTreeNode node = (
                    DefaultMutableTreeNode) model.getObject();
                if (node.isRoot()) {
                    return new CategoryRootPanel(nodeViewModel, nodeView);
                } else {
                    Category category = (Category) node.getUserObject();
                    nodeViewModel.setEntity(category);
                    return new CategoryNodePanel(nodeViewModel, nodeView);
                }
            }
        };
        add(tree);
        tree.getTreeState().collapseAll();
    }

    private DefaultTreeModel convertToTreeModel(final List<Category> list) {
        DefaultTreeModel model = null;
        DefaultMutableTreeNode rootNode = new DefaultMutableTreeNode();
```

```
		add(rootNode, list);
		model = new DefaultTreeModel(rootNode);
		return model;

	}

	private void add(final DefaultMutableTreeNode parent,
			final List<Category> child) {
		for (Category category : child) {
			DefaultMutableTreeNode childNode = new DefaultMutableTreeNode(
					category);
			parent.add(childNode);
			List<Category> subCategoriesList = category.getCategories()
					.getApprovedCategories().getList();
			if (!subCategoriesList.isEmpty()) {
				add(childNode, subCategoriesList);
			}
		}
	}

}
```

The following is from the Wicket' Javadoc of the LinkTree class:

“Simple tree component that provides node panel with link allowing user to select individual nodes.”

The first argument of the constructor is the tree Wicket identifier. The second argument is obtained by the convertToTreeModel method that accepts a list of approved categories at the root level of the hierarchy of categories ans subcategories. This private method calls another private method to add to the root of the tree other nodes of the tree. This is accomplished by calling recursively the add method.

The protected newNodeComponent method of the LinkTree class is overridden to decide what each node will represent. If a node is the tree root the CategoryRootPanel class is used, otherwise an object of the CategoryNodePanel class is constructed.

The HTML code for the CategoryTreePanel component is rather simple:

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

<html xmlns:wicket>

<wicket:panel>

	<div class="content">
		<table style="border: 1px solid #eeeeee">
			<tr>
				<td>
					<div wicket:id="tree" class="my-tree">
					</div>
				</td>
			</tr>
			</table>
		</div>

</wicket:panel>

</html>
```

In the HTML code of the home page where the tree component is used, the tree.css reference is added,

```
	.treeNode {
		border: 1px solid silver;
		margin-top: 4px;
		padding-left:10px;
		min-width: 150px;
	}
```

and the table.css reference is put in comments to allow the use of the table CSS from Wicket:

```
	<!--
	<link rel="stylesheet" type="text/css" media="screen" href="css/table.css" />
	-->
	<link rel="stylesheet" type="text/css" media="screen" href="css/tree.css" />
```

The CategoryRootPanel class extends the Panel class. It has only the link to add a category at the root level of the tree.

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

import org.apache.wicket.markup.html.basic.Label;
import org.apache.wicket.markup.html.link.Link;
import org.apache.wicket.markup.html.panel.Panel;
import org.modelibra.wicket.concept.EntityAddFormPage;
import org.modelibra.wicket.util.LocalizedText;
import org.modelibra.wicket.view.View;
import org.modelibra.wicket.view.ViewModel;

import dmeduc.weblink.WebLink;

public class CategoryRootPanel extends Panel {

	public CategoryRootPanel(final ViewModel viewModel, final View view) {
		super(view.getWicketId());

		String categories = LocalizedText.getApplicationPropertiesText(this,
				"Categories");
		add(new Label("rootName", categories));

		WebLink webLink = (WebLink) viewModel.getModel();
		final ViewModel categoriesAddViewModel = new ViewModel();
		categoriesAddViewModel.copyPropertiesFrom(viewModel);
		categoriesAddViewModel.setEntities(webLink.getCategories());
```

```
        categoriesAddViewModel.setEntity(null);

        add(new Link("categoryAddLink") {
            public void onClick() {
                setResponsePage(new EntityAddFormPage(categoriesAddViewModel,
                        view));
            }
        });
    }

}
```

The CategoryRootPanel.html has the root name label and the category add link.

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

<html xmlns:wicket>

<wicket:panel>

    <span wicket:id="rootName"/>
    <a wicket:id="categoryAddLink">
        <img src="css/img/document-new.png" alt="Add"/>
    </a>

</wicket:panel>

</html>
```

The CategoryNodePanel class also extends the Panel class. It has three subcomponents: a link to the Category Urls page, a link to add a new subcategory of the node category, and a display list of urls.

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

import org.apache.wicket.markup.html.basic.Label;
import org.apache.wicket.markup.html.link.Link;
import org.apache.wicket.markup.html.panel.Panel;
import org.apache.wicket.model.Model;
import org.apache.wicket.model.PropertyModel;
import org.modelibra.wicket.concept.EntityAddFormPage;
import org.modelibra.wicket.view.View;
import org.modelibra.wicket.view.ViewModel;

import dmeduc.weblink.category.Category;
import dmeduc.wicket.weblink.url.UrlDisplayList;

public class CategoryNodePanel extends Panel {

    public CategoryNodePanel(final ViewModel viewModel, final View view) {
        super(view.getWicketId(), new Model(viewModel.getEntity()));
```

```
            Category category = (Category) viewModel.getEntity();

            Link categoryPageLink = CategoryUrlsPage.link("categoryPageLink",
                        viewModel, view);
            categoryPageLink.add(new Label("categoryName", new PropertyModel(
                        category, "name")));
            add(categoryPageLink);

            final ViewModel categoriesAddViewModel = new ViewModel();
            categoriesAddViewModel.copyPropertiesFrom(viewModel);
            categoriesAddViewModel.setEntities(category.getCategories());
            categoriesAddViewModel.setEntity(null);

            add(new Link("categoryAddLink") {
                  public void onClick() {
                        setResponsePage(new EntityAddFormPage(categoriesAddViewModel,
                                    view));
                  }
            });

            add(new UrlDisplayList("urlDisplayList", category));
      }

}
```

The HTML code of the CategoryNodePanel class has two category links and a display list of web links. Note that the HTML code for a list view is found in its container, which is here CategoryNodePanel.

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

<html xmlns:wicket>

<wicket:panel>

<div class = "treeNode">
      <a wicket:id="categoryPageLink">
            <span wicket:id="categoryName">categoryName</span>
      </a>
      <a wicket:id="categoryAddLink">
            <img src="css/img/document-new.png" alt="Add"/>
      </a>
      <ul>
            <li wicket:id = "urlDisplayList">
                  <a wicket:id = "urlLink"></a>
            </li>
      </ul>
</div>

</wicket:panel>

</html>
```

The UrlDisplayList class extends the ListView class from Wicket. It uses the property model for the category entity so that its approved urls ordered by name are synchronized with a displayed list of urls. The list contains only names of clickable web links.

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

import org.apache.wicket.markup.html.link.ExternalLink;
import org.apache.wicket.markup.html.list.ListItem;
import org.apache.wicket.markup.html.list.ListView;
import org.apache.wicket.model.PropertyModel;

import dmeduc.weblink.category.Category;
import dmeduc.weblink.url.Url;

public class UrlDisplayList extends ListView {


	public UrlDisplayList(String wicketId, Category category) {
		super(wicketId, new PropertyModel(category,"approvedUrlsOrderedByName"));
	}

	protected void populateItem(ListItem item) {
		Url url = (Url) item.getModelObject();
		item.add(new ExternalLink("urlLink", url.getLink(), url.getName()));
	}

}
```

Summary

By developing specific web components and specific web pages, the web application becomes more user friendly. After a member has signed in, he may enter his page by clicking on the *My Page* link. The page shows his interests. For each interest there is a *Category* link to show the approved category with its approved web links. The member may edit his interests and maintain the corresponding web links. There is a new link on the home page called *Question Selection*. A subset of questions may be selected based on keywords that appear in the *Text* field. The *Sign Up* page enables a new user to register. This time he will receive an email with the confirmation number that he will have to enter in order to become a regular member. This is an improvement with respect to the previous spiral, where a casual member was admitted without an email validation. Finally, a tree of categories may be expanded on the home page to see web links of a certain subcategory, no matter how deep that subcategory is in the hierarchy of categories. In addition a new subcategory may be added to a parent category somewhere in the tree.

The next section presents some advanced concepts. The next chapter will show how generic web components can be created based on the experience of developing specific web components.

Questions

1. Why the Applicant concept is used for a sign up instead of the Member concept?

2. Why a parent-child page is more user friendly than two pages, one for the parent and the other for the child?

3. Why generic web components in ModelibraWicket use Modelibra configurations?

4. Can a web page be a generic component?

Exercises

Exercise 11.1.

Provide an email for a new regular member so that he can click on the link within the email message to confirm his registration. This would be a standard way of confirmation instead of copying and pasting the confirmation number in the confirmation field.

Exercise 11.2.

Create the specific MyPage class to have three sections: the grand-parent section for the member, the parent section for the member interests and the child section for the selected interest category with its approved web links. All three sections should be updatable.

Exercise 11.3.

Create the specific QuestionSelection class to provide keyword selections based on both text and response properties.

Web Links

[Inner Class] Java Inner Classes
http://www.javaworld.com/javaworld/javaqa/2000-03/02-qa-innerclass.html