SECTION V: Advanced Concepts

Chapter 12: Generic Web Components

The objective of this chapter is to show how to generalize specific web components into generic web components that can be easily reused in different contexts. The corresponding spiral is DmEduc-10. Specific web components are useful when we are preoccupied by a web application at hand. By making them, we respond to immediate needs, but we also learn quite a lot. However, if a similar situation appears in another web application, we may be better off to invest our time in developing a generic web component that will be reused in more than one application. In addition, by sharing generic web components in the open source community, we all will be better off. The final objective of ModelibraWicket will be to maintain an open source catalog of generic web components, so that a new web page will simply require a decomposition into sections, where for each page section a generic web component would be hopefully available.

Domain Model

The Web Link model from the previous spiral has not been changed (Figure 12.1).

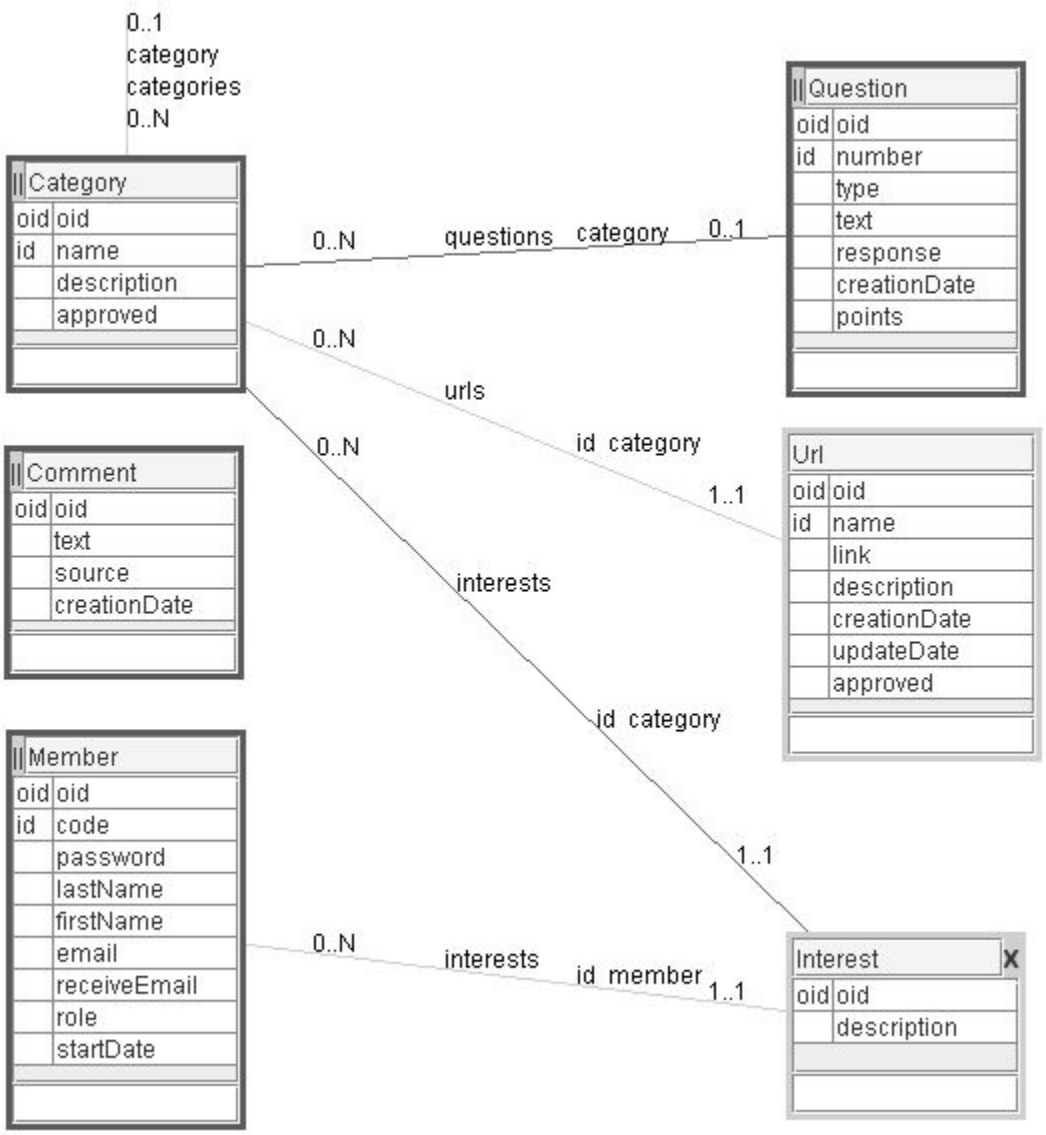

Figure 12.1. Web Link Model

Simple Component

In order to prepare a terrain for more complex generic web components, we will start with a simple component. There is a generic web component in ModelibraWicket that is called EntityDisplayMinPanel. For an entity, the component displays only essential properties. For each essential property there is a name and its value.

The component is placed in the org.modelibra.wicket.concept package where all generic web components related to a model concept are regrouped. The component extends the DmPanel container component. As any other generic web component in ModelibraWicket, the component has two generic arguments, the first one for the view model and the second one for the view. The DmPanel container component inherits its properties from the Panel component from Wicket, which requires a Wicket id in its constructor. Thus, the Wicket id is passed to the inheritance parent by using the super keyword. The

component is constructed in the try catch block.

```
package org.modelibra.wicket.concept;

import java.util.ArrayList;
import java.util.List;

import org.apache.wicket.markup.html.basic.Label;
import org.apache.wicket.markup.html.list.ListView;
import org.apache.wicket.markup.html.panel.Panel;
import org.modelibra.IEntity;
import org.modelibra.config.ConceptConfig;
import org.modelibra.config.PropertiesConfig;
import org.modelibra.config.PropertyConfig;
import org.modelibra.type.PropertyClass;
import org.modelibra.type.ValidationType;
import org.modelibra.wicket.app.DomainApp;
import org.modelibra.wicket.container.DmPanel;
import org.modelibra.wicket.container.PropertyNameLabelValuePanelListView;
import org.modelibra.wicket.util.LocalizedText;
import org.modelibra.wicket.util.PropertyNameLabelValuePanelPair;
import org.modelibra.wicket.view.View;
import org.modelibra.wicket.view.ViewModel;
import org.modelibra.wicket.widget.CheckBoxPanel;
import org.modelibra.wicket.widget.ExternalLinkPanel;
import org.modelibra.wicket.widget.LabelPanel;
import org.modelibra.wicket.widget.MultiLineLabelPanel;

public class EntityDisplayMinPanel extends DmPanel {

	public EntityDisplayMinPanel(final ViewModel viewModel, final View view) {
		super(view.getWicketId());
		DomainApp app = (DomainApp) getApplication();
		IEntity<?> entity = viewModel.getEntity();
		ConceptConfig conceptConfig = entity.getConceptConfig();

		List<PropertyNameLabelValuePanelPair> propertyNameLabelValuePanelPairs =
			new ArrayList<PropertyNameLabelValuePanelPair>();

		ViewModel propertyModel = new ViewModel();
		propertyModel.copyPropertiesFrom(viewModel);
		propertyModel.setEntity(entity);

		PropertiesConfig propertiesConfig = conceptConfig.getPropertiesConfig();
		for (PropertyConfig propertyConfig : propertiesConfig) {
			if (propertyConfig.isEssential()) {
				String propertyName = LocalizedText.getPropertyName(this,
						entity, propertyConfig);
				PropertyNameLabelValuePanelPair propertyNameLabelValuePanelPair
				= new PropertyNameLabelValuePanelPair();
				Label propertyNameLabel = new Label("propertyName",
						propertyName);
```

```
            propertyNameLabelValuePanelPair
                        .setPropertyNameLabel(propertyNameLabel);

            propertyModel.setPropertyConfig(propertyConfig);
            View propertyValueView = new View();
            propertyValueView.copyPropertiesFrom(view);
            propertyValueView.setWicketId("valuePanel");
            Panel essentialPropertyPanel;
            if (propertyConfig.getPropertyClass().equals(
                        PropertyClass.getUrl())
                        || propertyConfig.getPropertyClass().equals(
                                    PropertyClass.getEmail())) {
                essentialPropertyPanel = new ExternalLinkPanel(
                            propertyModel, propertyValueView);
            } else if (propertyConfig.getPropertyClass().equals(
                        PropertyClass.getString())
                        && propertyConfig.isValidateType()
                        && (propertyConfig.getValidationType().equals(
                                    ValidationType.getUrl()) || propertyConfig
                                    .getValidationType().equals(
                                                ValidationType.getEmail()))) {
                essentialPropertyPanel = new ExternalLinkPanel(
                            propertyModel, propertyValueView);
            } else if (propertyConfig.getPropertyClass().equals(
                        PropertyClass.getBoolean())) {
                essentialPropertyPanel = new CheckBoxPanel(propertyModel,
                            propertyValueView);
            } else if (propertyConfig.getPropertyClass().equals(
                        PropertyClass.getString())
                        && propertyConfig.getDisplayLengthInt() >
                    DomainApp.MIN_LONG_TEXT_LENGTH) {
                    essentialPropertyPanel = new MultiLineLabelPanel(
                            propertyModel, propertyValueView);
            } else {
                    essentialPropertyPanel = new LabelPanel(propertyModel,
                            propertyValueView);
            }
            if (!app.getAccessPoint().isPropertyDisplayAllowed(
                        getAppSession(), propertyConfig)) {
                    essentialPropertyPanel.setVisible(false);
            }

            propertyNameLabelValuePanelPair
                        .setPropertyValuePanel(essentialPropertyPanel);
            propertyNameLabelValuePanelPairs
                        .add(propertyNameLabelValuePanelPair);
      } // if (propertyConfig.isEssential()) {
} // end for

ListView propertyNameLabelValuePanelListView = new
      PropertyNameLabelValuePanelListView(
            "propertyNameLabelValuePanelListView",
```

```
                        propertyNameLabelValuePanelPairs);
            add(propertyNameLabelValuePanelListView);
            if (!app.getAccessPoint().isConceptDisplayAllowed(getAppSession(),
                        conceptConfig)) {
                  propertyNameLabelValuePanelListView.setVisible(false);
            }
      }

}
```

The web application is obtained by the inherited getApplication method. Since in ModelibraWicket a specific web application, such as DmEducApp, extends the DomainApp class, the casting is done to convert a Wicket web application into a ModelibraWicket application. The entity in question is found in the view model argument. The concept configuration is obtained from the entity object.

```
DomainApp app = (DomainApp) getApplication();
IEntity entity = viewModel.getEntity();
ConceptConfig conceptConfig = entity.getConceptConfig();
```

The Java List class is used to prepare a list of property pairs. A property pair consists of a property name label and a property panel value. This means that a property name will be shown as a label widget and a property value will be displayed as a panel to generalize different types of values that a property may have.

```
List<PropertyNameLabelValuePanelPair>
      propertyNameLabelValuePanelPairs = new
      ArrayList<PropertyNameLabelValuePanelPair>();
```

The PropertyNameLabelValuePanelPair class is a simple POJO [POJO].

```
package org.modelibra.wicket.util;

import java.io.Serializable;

import org.apache.wicket.markup.html.basic.Label;
import org.apache.wicket.markup.html.panel.Panel;

public class PropertyNameLabelValuePanelPair implements Serializable {

      private Label propertyNameLabel;

      private Panel propertyValuePanel;

      public PropertyNameLabelValuePanelPair() {
            super();
      }

      public void setPropertyNameLabel(Label propertyNameLabel) {
            this.propertyNameLabel = propertyNameLabel;
      }
```

```
	public Label getPropertyNameLabel() {
		return propertyNameLabel;
	}

	public void setPropertyValuePanel(Panel propertyValuePanel) {
		this.propertyValuePanel = propertyValuePanel;
	}

	public Panel getPropertyValuePanel() {
		return propertyValuePanel;
	}

}
```

The EntityDisplayMinPanel generic web component has its own context defined in the two arguments. This context is used as the starting point to define a new model context for a subcomponent of the component. This subcomponent is a widget from ModelibraWicket called LabelPanel. Actually, the component has two subcomponents, one for the property name label and another for the property value panel. The Wicket's Label will be used for the property name and the ModelibraWicket's LabelPanel will be used for the property value.

```
ViewModel propertyModel = new ViewModel();
propertyModel.copyPropertiesFrom(viewModel);
propertyModel.setEntity(entity);
```

From the concept configuration, the property configuration entities are reached and in the **for** iteration only an essential property is considered.

```
PropertiesConfig propertiesConfig = conceptConfig
		.getPropertiesConfig();
for (PropertyConfig propertyConfig : propertiesConfig) {
	if (propertyConfig.isEssential()) {
		...
	}
}
```

The property name in the current natural language is found with the help of the LocalizedText class.

```
	public static String getPropertyName(Component comp, IEntity<?> entity,
			PropertyConfig propertyConfig) {
		String propertyKey = entity.getConceptConfig().getCode() + "."
				+ propertyConfig.getCode();

		return LocalizedText.getApplicationPropertiesText(comp, propertyKey);
	}
```

The Label widget from Wicket is used to create the property name with the corresponding Wicket id.

```
String propertyName = LocalizedText.getPropertyName(this,
		entity, propertyConfig);
PropertyNameLabelValuePanelPair
```

```
        propertyNameLabelValuePanelPair = new
PropertyNameLabelValuePanelPair();
Label propertyNameLabel = new Label("propertyName",
            propertyName);
propertyNameLabelValuePanelPair
            .setPropertyNameLabel(propertyNameLabel);
```

The property model is informed about the property configuration. Then, the property value view is prepared from the view argument of the component. The essential property panel from Wicket is used to accept different types of panels from ModelibraWicket. Thus, in the first case of a property configuration that is either a url or an email, the ExternalLinkPanel widget from ModelibraWicket is used.

```
propertyModel.setPropertyConfig(propertyConfig);
View propertyValueView = new View();
propertyValueView.copyPropertiesFrom(view);
propertyValueView.setWicketId("valuePanel");
Panel essentialPropertyPanel;
if (propertyConfig.getPropertyClass().equals(
            PropertyClass.getUrl())
            || propertyConfig.getPropertyClass().equals(
                        PropertyClass.getEmail())) {
      essentialPropertyPanel = new ExternalLinkPanel(
                  propertyModel, propertyValueView);
}
```

If a property value is of the String type but it is still clickable, the same ExternalLinkPanel subcomponent is used.

```
} else if (propertyConfig.getPropertyClass().equals(
            PropertyClass.getString())
            && propertyConfig.isValidateType()
            && (propertyConfig.getValidationType().equals(
                        ValidationType.getUrl()) ||
                        propertyConfig
                              .getValidationType().equals(
                              ValidationType.getEmail()))) {
      essentialPropertyPanel = new ExternalLinkPanel(
                  propertyModel, propertyValueView);
}
```

If a property value is of the Boolean type, its value is created by the constructor of the CheckBoxPanel class.

```
} else if (propertyConfig.getPropertyClass().equals(
            PropertyClass.getBoolean())) {
      essentialPropertyPanel = new
            CheckBoxPanel(propertyModel, propertyValueView);
}
```

If a property value is of the String type and if its value length is longer than a predefined constant, the

value is displayed with a help from the MultiLineLabelPanel widget from ModelibraWicket.

```
} else if (propertyConfig.getPropertyClass().equals(
        PropertyClass.getString())
        && propertyConfig.getDisplayLengthInt() >
        DomainApp.MIN_LONG_TEXT_LENGTH) {
    essentialPropertyPanel = new MultiLineLabelPanel(
            propertyModel, propertyValueView);
}
```

In any other case, the property value will be displayed as an ordinary label, but by using the LabelPanel from ModelibraWicket.

```
} else {
    essentialPropertyPanel = new LabelPanel(propertyModel,
            propertyValueView);
}
```

The essential property panel will be made invisible if a user does not have a permission to see the property value.

```
if (!app.getAccessPoint().isPropertyDisplayAllowed(
        getAppSession(), propertyConfig)) {
    essentialPropertyPanel.setVisible(false);
}
```

The essential property panel becomes the second value of the pair object (the first has been the property name label). Finally, the pair is added to a list of pairs and the **for** iteration continues for all essential properties.

```
propertyNameLabelValuePanelPair
        .setPropertyValuePanel(essentialPropertyPanel);
propertyNameLabelValuePanelPairs
        .add(propertyNameLabelValuePanelPair);
```

After all essential properties are processed, the ListView component from Wicket is fed with our list of pairs. The list view will be hidden if the display of the concept is not allowed.

```
    ListView propertyNameLabelValuePanelListView = new
PropertyNameLabelValuePanelListView(
        "propertyNameLabelValuePanelListView",
        propertyNameLabelValuePanelPairs);
    add(propertyNameLabelValuePanelListView);
    if (!app.getAccessPoint().isConceptDisplayAllowed(getAppSession(),
            conceptConfig)) {
        propertyNameLabelValuePanelListView.setVisible(false);
    }
```

Thanks to Wicket, the HTML code for the EntityDisplayMinPanel generic web component is rather simple.

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

<html xmlns:wicket>

<wicket:panel>

  <table>
       <tr wicket:id = "propertyNameLabelValuePanelListView">
            <th wicket:id = "propertyName" align = "right">
                Property name
            </th>
            <td wicket:id = "valuePanel">
                Value panel
          </td>
       </tr>
  </table>

  <wicket:child/>

</wicket:panel>

</html>
```

Generic Tree Component

The home page uses a generic tree component for the tree of categories. The generic web component is called AjaxEntitiyTreePanel. It requires two specific panels that a developer must provide. The first panel is for the root node. The second one is for a node panel. The root and node panels are represented here by the CategoryRootPanel and CategoryNodePanel classes respectively. This mixture of the generic component for a tree and specific components for tree nodes provides both productivity and flexibility.

```
     add(new AjaxEntitiyTreePanel(treeViewModel, treeView) {

          @Override
          protected Panel getRootNodePanel(ViewModel viewModel, View view) {
               return new CategoryRootPanel(viewModel, view);
          }

          @Override
          protected Panel getNodePanel(ViewModel viewModel, View view) {
               return new CategoryNodePanel(viewModel, view);
          }
     });
```

The generic tree component extends the DmPanel container component. It has the standard ModelibraWicket constructor, two private methods and two protected methods. The constructor finds entities from the view model and checks if there is a neighbor with the same configuration. If yes, the

relationship is reflexive and a tree can be built. The tree is built from the list of entities with a help of the private convertToTreeModel method. This method creates the root node and adds the list of child nodes to the root node by the other private add method. The add method is recursive. After a child node based on an entity from the list is added to the root node, its sub-entities are used as the basis to add grandchild nodes to the child node, etc. The protected newNodeComponent method of the LinkTree class is called by Wicket for each new node of the tree. If the new node is the root node, the protected getRootNodePanel method would be called. If the new node is not the root node, the protected getNodePanel method would be called.

```
package org.modelibra.wicket.neighbor.tree;

import java.util.Iterator;
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.IEntities;
import org.modelibra.IEntity;
import org.modelibra.config.ConceptConfig;
import org.modelibra.config.NeighborConfig;
import org.modelibra.config.NeighborsConfig;
import org.modelibra.exception.ModelibraRuntimeException;
import org.modelibra.wicket.concept.EntitiesNameLabelAddLinkPanel;
import org.modelibra.wicket.concept.EntityDisplayAddLinksPanel;
import org.modelibra.wicket.container.DmPanel;
import org.modelibra.wicket.view.View;
import org.modelibra.wicket.view.ViewModel;

public class AjaxEntityTreePanel extends DmPanel {

	private String code;

	public AjaxEntityTreePanel(final ViewModel viewModel, final View view) {
		super(view.getWicketId());
		final IEntities<?> entities = viewModel.getEntities();
		ConceptConfig conceptConfig = entities.getConceptConfig();
		code = conceptConfig.getEntitiesCodeWithFirstLetterAsLower();

		NeighborsConfig neighborsConfig = conceptConfig.getNeighborsConfig();

		boolean hasReflexiveRelationship = false;
		for (NeighborConfig neighborConfig : neighborsConfig) {
			if (neighborConfig.getDestinationConceptConfig() == conceptConfig
					&& neighborConfig.isChild()) {
				hasReflexiveRelationship = true;
				code = neighborConfig.getCodeWithFirstLetterAsLower();
```

```
                    break;
               }

          }
          if (hasReflexiveRelationship) {
               List<IEntity> entitiesList = (List<IEntity>) entities.getList();

               LinkTree tree = new LinkTree("tree",
                         convertToTreeModel(entitiesList)) {

                    private static final long serialVersionUID = 1L;

                    @Override
                    protected Component newNodeComponent(String id, IModel model) {

                         View nodeView = new View();
                         nodeView.setPage(getPage());
                         nodeView.setWicketId(id);
                         nodeView.setContextView(nodeView);
                         nodeView.setRecreateContext(true);

                         ViewModel nodeViewModel = new ViewModel();
                         nodeViewModel.copyPropertiesFrom(viewModel);
                         nodeViewModel.setEntities(entities);
                         nodeViewModel.setContextViewModel(viewModel);

                         DefaultMutableTreeNode node = (DefaultMutableTreeNode) model
                                   .getObject();
                         if (node.isRoot()) {
                              return getRootNodePanel(nodeViewModel, nodeView);
                         } else {
                              IEntity entity = (IEntity) node.getUserObject();
                              nodeViewModel.setEntity(entity);
                              return getNodePanel(nodeViewModel, nodeView);
                         }
                    }
               };
               add(tree);
               tree.getTreeState().collapseAll();
          } else {
               throw new ModelibraRuntimeException(
                    "EntitiesTreePanel can be used only with entities of a
                         concept that has reflexive relationship!");
          }
     }

     private DefaultTreeModel convertToTreeModel(List<IEntity> list) {
          DefaultTreeModel model = null;
          DefaultMutableTreeNode rootNode = new DefaultMutableTreeNode();
          add(rootNode, list);
          model = new DefaultTreeModel(rootNode);
          return model;
```

```
	}

	private void add(DefaultMutableTreeNode parent, List<IEntity> sub) {
		Iterator<IEntity> i = sub.iterator();
		for (IEntity entity : sub) {
			DefaultMutableTreeNode child = new DefaultMutableTreeNode(entity);
			parent.add(child);
			List<IEntity> subEntitiesList = entity.getChildNeighbor(code)
					.getList();
			if (!subEntitiesList.isEmpty()) {
				add(child, subEntitiesList);
			}
		}
	}

	protected Panel getRootNodePanel(ViewModel viewModel, View view) {
		return new EntitiesNameLabelAddLinkPanel(viewModel, view);
	}

	protected Panel getNodePanel(ViewModel viewModel, View view) {
		return new EntityDisplayAddLinksPanel(viewModel, view);
	}

}
```

If a developer does not override the protected getRootNodePanel method, a panel constructed with the EntitiesNameLabelAddLinkPanel will be returned.

```
package org.modelibra.wicket.concept;

import org.apache.wicket.markup.html.basic.Label;
import org.apache.wicket.markup.html.link.Link;
import org.modelibra.IEntities;
import org.modelibra.wicket.app.DomainApp;
import org.modelibra.wicket.container.DmPanel;
import org.modelibra.wicket.util.LocalizedText;
import org.modelibra.wicket.view.View;
import org.modelibra.wicket.view.ViewModel;

public class EntitiesNameLabelAddLinkPanel extends DmPanel {

	public EntitiesNameLabelAddLinkPanel(final ViewModel viewModel, final View
			view) {
		super(view.getWicketId());
		final DomainApp domainApp = (DomainApp) getApplication();
		final String modelCode = viewModel.getModel().getModelConfig()
				.getCode();

		// root node label
		IEntities<?> entities = viewModel.getEntities();
		String conceptsName = entities.getConceptConfig().getConceptsName();
```

```
            String localizedConceptsName = LocalizedText
                        .getApplicationPropertiesText(this, conceptsName);
            add(new Label("rootName", localizedConceptsName));

            // Entity add link
            final ViewModel entityAddViewModel = new ViewModel();
            entityAddViewModel.copyPropertiesFrom(viewModel);
            entityAddViewModel.setEntities(entities);
            entityAddViewModel.setEntity(null);

            add(new Link("entityAddLink") {
                  public void onClick() {
                        setResponsePage(domainApp.getViewMeta(modelCode).getPage(
                              "EntityAddFormPage", entityAddViewModel, view));
                  }
            });
      }

}
```

If a developer does not override the protected getNodePanel method, a panel constructed with the EntityDisplayAddLinksPanel will be returned.

```
package org.modelibra.wicket.concept;

import org.apache.wicket.markup.html.basic.Label;
import org.apache.wicket.markup.html.link.Link;
import org.apache.wicket.model.PropertyModel;
import org.modelibra.IEntity;
import org.modelibra.wicket.app.DomainApp;
import org.modelibra.wicket.container.DmPanel;
import org.modelibra.wicket.view.View;
import org.modelibra.wicket.view.ViewModel;

public class EntityDisplayAddLinksPanel extends DmPanel {

      public EntityDisplayAddLinksPanel(final ViewModel viewModel, final View view) {
            super(view.getWicketId());
            final DomainApp domainApp = (DomainApp) getApplication();
            final String modelCode = viewModel.getModel().getModelConfig()
                        .getCode();

            IEntity<?> entity = (IEntity<?>) viewModel.getEntity();

            // EntityDisplayPage link
            Link entityDisplayPageLink = new Link("entityDisplayPageLink") {
                  @Override
                  public void onClick() {
                        setResponsePage(domainApp.getViewMeta(modelCode).getPage(
                                    "EntityDisplayPage", viewModel, view));
                  }
            };
```

```
            entityDisplayPageLink.add(new Label("entityString", new PropertyModel(
                        entity, "toString()")));
            add(entityDisplayPageLink);

            // EntityAddFormPage link
            String code = entity.getConceptConfig().getEntitiesCodeInLowerLetters();
            final ViewModel categoriesAddViewModel = new ViewModel();
            categoriesAddViewModel.copyPropertiesFrom(viewModel);
            categoriesAddViewModel.setEntities(entity.getChildNeighbor(code));
            categoriesAddViewModel.setEntity(null);

            add(new Link("entityAddLink") {
                  public void onClick() {
                        setResponsePage(domainApp.getViewMeta(modelCode).getPage(
                                    "EntityAddFormPage", viewModel, view));
                  }
            });
      }

}
```

Parent Child Component

The link to *My Page* in the home menu is replaced by the link to the parent child generic component.

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

The name of the generic component is EntityParentChildUpdatePage.

```
package org.modelibra.wicket.neighbor;

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.container.DmUpdatePage;
```

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

public class EntityParentChildUpdatePage extends DmUpdatePage {

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

		View parentChildPageView = new View();
		parentChildPageView.copyPropertiesFrom(view);
		parentChildPageView.setWicketId("parentChildUpdateSection");
		parentChildPageView.setContextView(view);
		parentChildPageView.setPage(this);

		add(new EntityParentChildUpdatePanel(parentChildPageModel,
				parentChildPageView));
	}

	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 EntityParentChildUpdatePage(viewModel, view);
			}

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

}
```

The page has only one section which is also generic: EntityParentChildUpdatePanel. The panel component extends the EntityParentChildPanel abstract class. There are two protected methods that implement corresponding abstract methods in the inheritance parent. In order to allow the parent update, the EntityEditFormPanel component is used in the getParentPanel method. Similarly, the EntityUpdateTablePanel component is used in the getChildPanel method.

```
package org.modelibra.wicket.neighbor;

import org.apache.wicket.markup.html.panel.Panel;
import org.modelibra.wicket.app.DomainApp;
import org.modelibra.wicket.view.View;
import org.modelibra.wicket.view.ViewModel;

public class EntityParentChildUpdatePanel extends EntityParentChildPanel {
```

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

	protected Panel getParentPanel(ViewModel viewModel, View view) {
		DomainApp app = (DomainApp) getApplication();
		String model = viewModel.getModel().getModelConfig().getCode();
		return app.getViewMeta(model).getPanel("EntityEditFormPanel",
				viewModel, view);
	}

	protected Panel getChildPanel(ViewModel viewModel, View view) {
		DomainApp app = (DomainApp) getApplication();
		String model = viewModel.getModel().getModelConfig().getCode();
		return app.getViewMeta(model).getPanel("EntityUpdateTablePanel",
				viewModel, view);
	}

}
```

The EntityParentChildPanel abstract class provides context for the parent and the child subcomponents. By default, the first child for the parent is retrieved by the getChildEntities method. Of course, this method is protected to allow a change of the child entities by a subclass.

```
package org.modelibra.wicket.neighbor;

import java.util.List;

import org.apache.wicket.markup.html.panel.EmptyPanel;
import org.apache.wicket.markup.html.panel.Panel;
import org.modelibra.IEntities;
import org.modelibra.IEntity;
import org.modelibra.config.ConceptConfig;
import org.modelibra.wicket.container.DmPanel;
import org.modelibra.wicket.view.View;
import org.modelibra.wicket.view.ViewModel;

public abstract class EntityParentChildPanel extends DmPanel {

	public EntityParentChildPanel(final ViewModel viewModel, final View view) {
		super(viewModel, view);
		// Parent
		ViewModel parentModel = new ViewModel();
		parentModel.copyPropertiesFrom(viewModel);

		View parentView = new View();
		parentView.copyPropertiesFrom(view);
		parentView.setWicketId("parentSection");

		add(getParentPanel(parentModel, parentView));
```

```
            // Child
            IEntity<?> parentEntity = viewModel.getEntity();
            IEntities<?> childEntities = getChildEntities(parentEntity);
            Panel childPanel;
            if (childEntities != null) {
                  ViewModel childModel = new ViewModel();
                  childModel.copyPropertiesFrom(viewModel);
                  childModel.setEntities(childEntities);
                  childModel.setEntity(null);

                  View childView = new View();
                  childView.copyPropertiesFrom(view);
                  childView.setContextView(view);
                  childView.setWicketId("childSection");

                  childPanel = getChildPanel(childModel, childView);
            } else {
                  childPanel = new EmptyPanel("childSection");
                  childPanel.setVisible(false);
            }
            add(childPanel);
      }

      protected abstract Panel getParentPanel(ViewModel viewModel, View view);

      protected abstract Panel getChildPanel(ViewModel viewModel, View view);

      protected IEntities<?> getChildEntities(IEntity<?> parentEntity) {
            ConceptConfig conceptConfig = parentEntity.getConceptConfig();
            List<String> childCodes = conceptConfig.getChildNeighborCodes();
            if (childCodes.size() > 0) {
                  return parentEntity.getChildNeighbor(childCodes.get(0));
            }
            return null;
      }

}
```

There are also EntityParentChildDisplayPage and EntityParentChildDisplayPanel generic display components that are similar to the generic update components. EntityParentChildDisplayPage contains EntityParentChildDisplayPanel that uses EntityDisplayPanel for parent entity display and EntityDisplayTablePanel for child entities display.

Category Urls

The CategoryUrlsPage class has been redeveloped by using the previously seen EntityParentChildUpdatePanel 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.IEntities;
import org.modelibra.IEntity;
import org.modelibra.wicket.container.DmUpdatePage;
import org.modelibra.wicket.neighbor.EntityParentChildUpdatePanel;
import org.modelibra.wicket.view.View;
import org.modelibra.wicket.view.ViewModel;

import dmeduc.weblink.category.Category;

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.setWicketId("categoryUrlsSection");
		categoryUrlsPageView.setPage(this);
		setVersioned(true);

		Panel categoryUrlsPanel = new EntityParentChildUpdatePanel(
				categoryUrlsPageModel, categoryUrlsPageView) {
			@Override
			protected IEntities<?> getChildEntities(IEntity<?>
				parentEntity) {
				Category category = (Category) parentEntity;
				if (category.isApproved()) {
					return category.getUrls();
				}
				return null;
			}
		};
		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 page class uses anonymous inner class that extends the EntityParentChildUpdatePanel component to provide child entities, but only if the current category is approved.

Generic Selection

The QuestionSelectionPage class shows how easy it is to use the PropertySelectorPanel generic web component. The component is added to the page and the getNewPageInstance method is overridden to provide a new instance of the page to get the effect of narrowing the selection. The context link of the page returns to the previous selection. All components that use the page's ViewModel will reflect the selection (in case of QuestionSelectionPage that is EntityDisplayTablePanel). The getNewPageInstance method is not abstract, and default response page is EntityDisplayTablePage.

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

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.EntityDisplayTablePanel;
import org.modelibra.wicket.concept.selection.PropertySelectorPanel;
import org.modelibra.wicket.container.DmDisplayPage;
import org.modelibra.wicket.view.View;
import org.modelibra.wicket.view.ViewModel;

public class QuestionSelectionPage extends DmDisplayPage {

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

            View contextView = new View();
            contextView.copyPropertiesFrom(view);
            contextView.setPage(this);

            View questionSelectionPanelView = new View();
            questionSelectionPanelView.copyPropertiesFrom(view);
            questionSelectionPanelView.setContextView(view);
            questionSelectionPanelView.setWicketId("selectionSection");
            setVersioned(true);
            PropertySelectorPanel selectionPanel = new PropertySelectorPanel(
                        questionSelectionPageModel, questionSelectionPanelView) {
                  @Override
```

```
                protected Page getNewPageInstance(ViewModel viewModel, View
                        view) {
                    return new QuestionSelectionPage(viewModel, view);
                }
            };
            add(selectionPanel);

            View entityDisplayTablePanelView = new View();
            entityDisplayTablePanelView.copyPropertiesFrom(view);
            entityDisplayTablePanelView.setWicketId("questions");
            entityDisplayTablePanelView.setContextView(contextView);
            entityDisplayTablePanelView.setPage(this);
            entityDisplayTablePanelView.setUpdate(false);

            EntityDisplayTablePanel questionsPanel = new
                EntityDisplayTablePanel(
                        questionSelectionPageModel, entityDisplayTablePanelView);
            add(questionsPanel);
        }

        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 QuestionSelectionPage(viewModel, view);
                }

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

}
```

The PropertySelectorPanel class and its supporting PropertySelectorBean class will not be explained in this chapter. This is left to advanced readers that may want to consult the source code of the ModelibraWicket project.

Sign Up Confirmation

A new user may sign up for a membership by clicking on the *Sign Up* link in the home page. In the SignUpPage class, the EntityAddFormPanel generic web component is added to the page.

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

                ViewModel signUpModel = new ViewModel(webLink);
```

```
        Applicants applicants = webLink.getApplicants();
        signUpModel.setEntities(applicants);
        signUpModel.getUserProperties().addUserProperty("appContextPath",
                    getAppContextPath());

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

The appContextPath user property is added to the view model. Its value is obtained by the getAppContextPath method.

```
private String getAppContextPath() {
      String appContextPath = "";

      HttpServletRequest req = getWebRequestCycle().getWebRequest()
                  .getHttpServletRequest();

      String scheme = req.getScheme(); // http
      String serverName = req.getServerName(); // localhost
      int serverPort = req.getServerPort(); // 8081
      String contextPath = req.getContextPath(); // /ModelibraWicketApp
      String servletPath = req.getServletPath(); // /app

      // i.e http://localhost:8081/ModelibraWicketApp/app/
      appContextPath += scheme + "://" + serverName + ":" + serverPort
                  + contextPath + servletPath;
      return appContextPath;
}
```

The generic EntityAddFormPanel component uses the generic EntityAddForm subcomponent. However, if there is a web component with the same name in the specific package for the concept at hand, the specific web subcomponent will be used by ModelibraWicket instead of the generic subcomponent. In our case, the concept is Applicant and the specific EntityAddForm subcomponent exists in the dmeduc.wicket.weblink.applicant package. In that specific subcomponent, which carries the generic name, the form submit button will send a confirmation email to the email address entered by the applicant.

```
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);
            signOutGest();
            setResponsePage(HomePage.class);
      }
}
```

The email is sent by the sendEmailToConfirm method located in the specific subcomponent.

```
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();

	String confirmationLink = getConfirmationLink(viewModel);

	applicant.emailMessage(emailConfig, messageSubject, messageStart + " "
			+ confirmationLink);
}
```

The email configuration from the email-config.xml file must be complete and valid in order to send an email.

The subject of the sent email may resemble the following text:

To confirm your application...

The message of the email may resemble the following text:

...please follow this link:
http://localhost:8081/ModelibraWicket/app/confirmation/register/1197992340990/ .

In the DmEducApp class from the dmeduc.wicket.app package, there is a specific code that instructs Wicket to produce a user friendly URL for this email.

```
@Override
protected void init() {
	super.init();
	mountBookmarkablePage("/confirmation", ConfirmationPage.class);
}
```

The register is a page parameter for the ConfirmationPage class and the provided number is its value.

The ConfirmationPage class is located in the dmeduc.wicket.weblink.applicant package. The class provides applicant entities and a component displayed if the confirmation is not valid.

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

import org.apache.wicket.Component;
```

```
import org.apache.wicket.PageParameters;
import org.modelibra.IEntities;
import org.modelibra.wicket.security.registration.RegistrationConfirmationPage;
import org.modelibra.wicket.view.View;
import org.modelibra.wicket.view.ViewModel;

import dmeduc.wicket.app.DmEducApp;

public class ConfirmationPage extends RegistrationConfirmationPage {

	public ConfirmationPage(PageParameters pageParameters) {
		super(pageParameters);
	}

	protected IEntities<?> getApplicantEntities() {
		DmEducApp dmEducApp = (DmEducApp) getApplication();
		return dmEducApp.getDmEduc().getWebLink().getApplicants();
	}

	protected Component getComponentForNotRegistered() {
		View view = new View();
		view.setWicketId("confirmation");
		return new ApplicantNotRegisteredPanel(new ViewModel(), view);
	}

}
```

The ApplicantNotRegisteredPanel specific component provides a link to the sign up page.

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

import org.apache.wicket.markup.html.link.PageLink;
import org.modelibra.wicket.container.DmPanel;
import org.modelibra.wicket.view.View;
import org.modelibra.wicket.view.ViewModel;

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

public class ApplicantNotRegisteredPanel extends DmPanel {

	public ApplicantNotRegisteredPanel(ViewModel viewModel, View view) {
		super(view.getWicketId());
		DmEducApp dmEducApp = (DmEducApp) getApplication();
		WebLink webLink = dmEducApp.getDmEduc().getWebLink();
		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);
      }

}
```

The ConfirmationPage class extends the RegistrationConfirmationPage generic web component. The class is abstract and it requires a subclass to implement the abstract getApplicantEntities method.

```
package org.modelibra.wicket.security.registration;

import java.util.List;

import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;
import org.apache.wicket.Component;
import org.apache.wicket.PageParameters;
import org.apache.wicket.util.string.StringValueConversionException;
import org.modelibra.IDomain;
import org.modelibra.IEntities;
import org.modelibra.IEntity;
import org.modelibra.IDomainModel;
import org.modelibra.IDomainModels;
import org.modelibra.DomainModel;
import org.modelibra.Oid;
import org.modelibra.config.DomainConfig;
import org.modelibra.exception.MetaException;
import org.modelibra.wicket.app.DomainApp;
import org.modelibra.wicket.container.DmPage;
import org.modelibra.wicket.view.View;
import org.modelibra.wicket.view.ViewModel;

public abstract class RegistrationConfirmationPage extends DmPage {

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

      public RegistrationConfirmationPage(PageParameters pageParameters) {
            try {
                  long uniqueNumber = pageParameters.getLong("register");
                  Oid oid = new Oid(uniqueNumber);

                  DomainApp domainApp = (DomainApp) getApplication();
                  IDomain domain = domainApp.getDomain();
                  DomainConfig domainConfig = domain.getDomainConfig();
                  String signinConcept = domainConfig.getSigninConcept();
                  DomainModel signinModel =
                        (DomainModel) getSigninModel(domain, signinConcept);
                  IEntities signinEntities = null;
                  IEntity<?> signinEntity = null;
                  if (signinModel != null) {
```

```
                signinEntities = signinModel.getEntry(signinConcept);
                if (signinEntities != null) {
                     signinEntity = ((DomainModel) signinModel).getModelMeta()
                               .createEntity(signinEntities);
                     if (signinEntity == null) {
                          log.error(signinModel
                          + " sign in model does not have the sign in entity.");
                     }
                } else {
                     log.error(signinModel
                     + " sign in model does not have the sign in entities.");
                }
           } else {
                log.error(domainConfig.getCode()
                + " domain does not have the sign in model.");
           }

           // Try to retrieve applicant based on page parameter "register/oid"
           IEntities applicants = getApplicantEntities();

           Class<?> applicantsClass = applicants.getClass();
           Class<?> signinEntitiesClass = signinEntities.getClass();
           if (applicantsClass.getSuperclass().equals(signinEntitiesClass)
                     || applicantsClass.equals(signinEntitiesClass)) {
                IEntity<?> applicant = applicants.retrieveByOid(oid);
                Component panel;
                if (applicant != null && signinEntity != null) {
                     signinModel.getModelMeta().updateProperties(signinEntity,
                               applicant);
                     if (signinEntities.add(signinEntity)) {
                          applicants.remove(applicant);
                     }
                     panel = getComponentForRegistered(applicant);
                } else {// there is no applicant
                     panel = getComponentForNotRegistered();
                }
                add(panel);
           } else {
                log.error("Error in RegistrationConfirmationPage: applicant
                     entities have to be subclass of signin concept: "
                     + signinConcept);
           }
      } catch (StringValueConversionException e) {
           // if user manually edits the url and break parameters redirect
           setResponsePage(getRedirectPageClass());
      } catch (MetaException e) {
           log.error("Error in " + getClass() + " :" + e.getMessage());
      }
}

protected abstract IEntities getApplicantEntities();
```

```
	protected Component getComponentForRegistered(IEntity<?> entity) {
		ViewModel viewModel = new ViewModel();
		viewModel.setEntity(entity);

		View view = new View();
		view.setWicketId("confirmation");
		return new RegisteredPanel(viewModel, view);
	};

	protected Component getComponentForNotRegistered() {
		View view = new View();
		view.setWicketId("confirmation");
		return new NotRegisteredPanel(new ViewModel(), view);
	};

	protected Class<?> getRedirectPageClass() {
		return getApplication().getHomePage();
	};

	private IDomainModel getSigninModel(IDomain domain, String signinConcept) {
		IDomainModel referenceModel = (DomainModel) domain.getReferenceModel();
		IEntities signinEntities = null;

		IDomainModels models = domain.getModels();
		List<IDomainModel> modelList = models.getList();
		for (IDomainModel model : modelList) {
			if (model.equals(referenceModel)) {
				continue;
			} else {
				signinEntities = model.getEntry(signinConcept);
				if (signinEntities != null) {
					return model;
				}
			}
		}

		if (referenceModel != null) {
			signinEntities = referenceModel.getEntry(signinConcept);
			if (signinEntities != null) {
				return referenceModel;
			}
		}

		return null;
	}

}
```

There are three protected methods in the class that can be overridden to provide specific components for informing a user, and to offer a different redirect page.

The confirmation is done by accepting a parameter that a user sent by clicking on the link in the

confirmation email. The parameter number is used to retrieve the applicant, to copy his values to the sign in entity, to add the sign in entity to its collection of entities, and finally to remove the applicant from its collection of entities.

There is no need to have the HTML template in the specific confirmation class. However, an additional content may be provided by the specific class and its HTML code that would extend the HTML code of the generic confirmation page by using the wicket:extend tag.

The generic component for informing an applicant that the registration was successful is called RegisteredPanel.

```
package org.modelibra.wicket.security.registration;

import org.apache.wicket.markup.html.basic.Label;
import org.apache.wicket.markup.html.link.Link;
import org.apache.wicket.model.IModel;
import org.apache.wicket.model.Model;
import org.apache.wicket.model.StringResourceModel;
import org.modelibra.IEntity;
import org.modelibra.wicket.container.DmPanel;
import org.modelibra.wicket.security.AccessPoint;
import org.modelibra.wicket.view.View;
import org.modelibra.wicket.view.ViewModel;

public class RegisteredPanel extends DmPanel {

	public RegisteredPanel(ViewModel viewModel, View view) {
		super(view.getWicketId());
		final IEntity<?> signinEntity = viewModel.getEntity();
		IModel entityModel = new Model(signinEntity);
		Label welvomeLabel = new Label("messageLabel", new StringResourceModel(
				"message", this, entityModel));
		add(welvomeLabel);

		Link siginLink = new Link("signinLink") {
			@Override
			public void onClick() {
				getAppSession().authenticate(signinEntity, AccessPoint.CODE,
						AccessPoint.PASSWORD);
				setResponsePage(getApplication().getHomePage());
			}
		};
		add(siginLink);
	}

}
```

The generic component for informing a user that the registration was not successful is named NotRegisteredPanel.

```
package org.modelibra.wicket.security.registration;
```

```
import org.apache.wicket.markup.html.link.PageLink;
import org.modelibra.wicket.container.DmPanel;
import org.modelibra.wicket.view.View;
import org.modelibra.wicket.view.ViewModel;

public class NotRegisteredPanel extends DmPanel {

	private static final long serialVersionUID = 1L;

	public NotRegisteredPanel(ViewModel viewModel, View view) {
		super(view.getWicketId());
		add(new PageLink("homePageLink", getApplication().getHomePage()));
	}

}
```

Summary

Specific web components are useful when we are preoccupied by a web application at hand. By making them, we respond to immediate needs, but we also learn quite a lot. However, if a similar situation appears in another web application, we may be better off to invest our time in developing a generic web component that will be reused in more than one application. In addition, by sharing generic web components in the open source community, we all will be better off.

Developing generic web components in Wicket by using Modelibra may look complex to beginners. However, there are not that many lines of code in a generic web component and the productivity gains in using generic web components as sections of specific web pages are pretty high.

The next chapter will focus on Ajax web components. Ajax is becoming popular because web pages that use Ajax look like windows on personal computers.

Questions

1. How would you characterize a generic web component?

2. What is the main difference between specific and generic components?

3. Is the tree component used in the home page an Ajax component?

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

Exercises

Exercise 12.1.

Develop a generic web component that will display two properties from the parent and two properties from the child in a parent-child relationship.

Exercise 12.2.

Consult the [Netvibes] and [Widgets] web sites and find a simple web component that you want to develop in ModelibraWicket.

Exercise 12.3.

Prepare a text document with a proposal for ten generic web components that would provide higher productivity in the (University) Course domain.

Web Links

[Generic] Generic Programming
http://www.cs.rpi.edu/~musser/gp/index.html

[Netvibes] Netvibes
http://www.netvibes.com/

[POJO] Plain Olda Java Object
http://en.wikipedia.org/wiki/POJO

[Widgets] Widgipedia
http://www.widgipedia.com/