SECTION IV: Web Components

Chapter 9: Web Components

The objective of this chapter is to create a specific home page by reusing generic components of ModelibraWicket and by creating specific web components by reusing generic components of Wicket. A web page is divided into sections. A section is presented through a web component. A web component may be specific or generic. A specific component is developed by a web application programmer with the objective to focus on a specific page section. A generic component is developed by an advanced programmer to support the needs of different web applications through its parameters. The first step in creating a generic web component of ModelibraWicket is the creation of the first specific web component with ModelibraWicket. The first step in creating a specific web component is the use of the first generic web component of ModelibraWicket.

A web component, specific or generic, uses a part of a domain model, called a component view model. A web component has also its view presentation parameters. A framework of the Modelibra software family that focuses on web components is ModelibraWicket. Of course, ModelibraWicket uses the Modelibra domain model framework for its web view models, and the Wicket web framework for its web views. In the opposite direction, Modelibra does not have any knowledge of ModelibraWicket, and Wicket does not know anything about ModelibraWicket and Modelibra.

The Wicket web framework is used extensively to develop specific web components and generic web components of ModelibraWicket. Thus, a good knowledge of Wicket is necessary for development of web components. However, there is no need to understand deeply Wicket in order to use either specific or generic web components.

Why is Wicket used in Modelibra for web components? The main reason is that, in Wicket, a development of web applications is concentrated in Java classes and not in other technologies. XHTML and CSS are used to display a web component view in a usual way. However, both a view model and a view are done completely in Java. It is hard to believe, but most web application developers spend quite some time in using technologies that are not object oriented.

Domain Model

The domain model from the previous spiral is used without changes in this chapter.

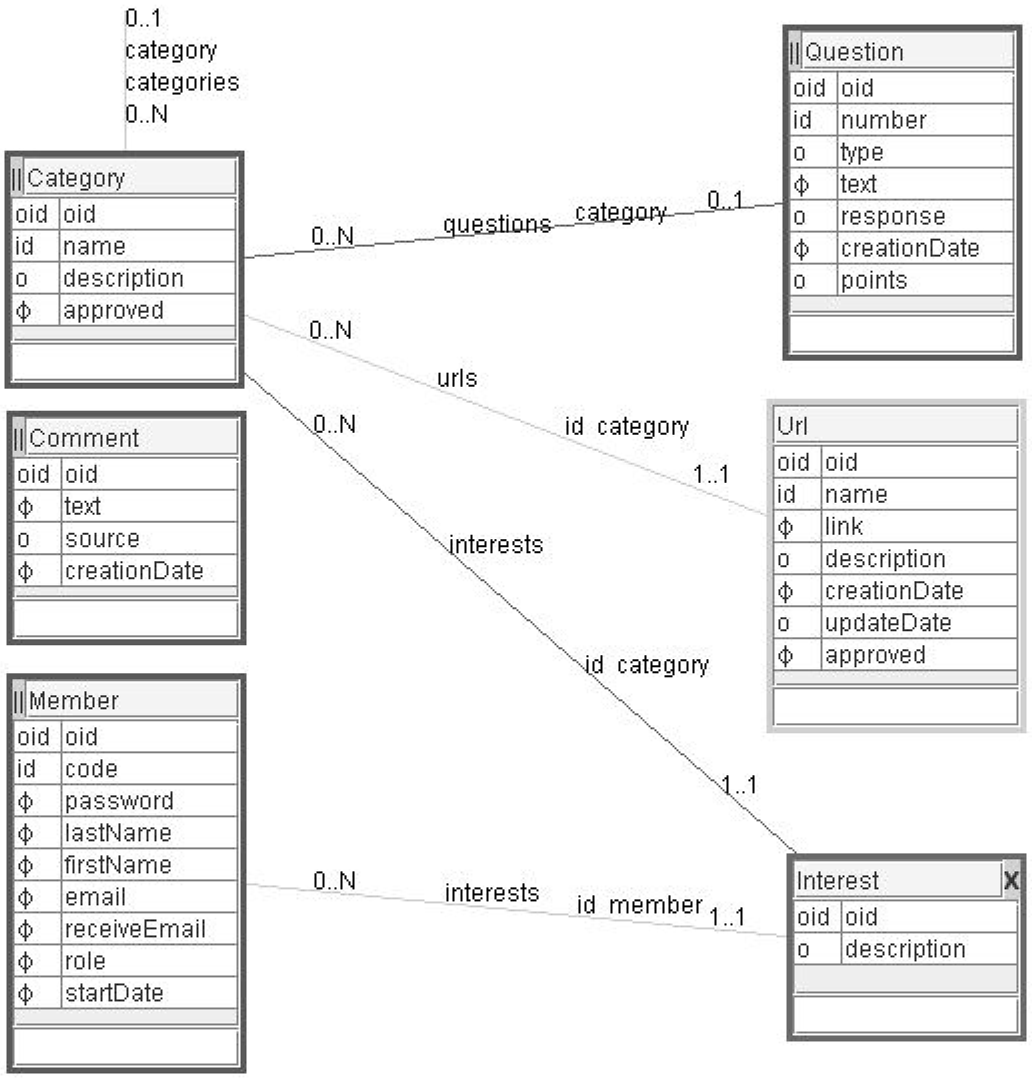


Figure 9.1. WebLink model

Home Page

Dynamic content of the application home page is determined by the HomePage specific class from the dmeduc.wicket.home package. The presentation of the home page is defined by the HomePage.html file that is located in the same package as its corresponding Java class. There is a tight coupling between a Java class and its .html file. They are both located in the same package and carry the same name. Of course, the file extensions are different: .java for a class and .html for its presentation file. The look of the home page is defined by CSS declarations defined in various .css files, which are referenced at the beginning of HomePage.html.

The home page is divided into three major areas (Figure 9.1). The three areas are positioned geographically into North, Middle and South. There are different ways to design a web page. Here, we will focus on a few standard ways to layout a page [Page layouts].

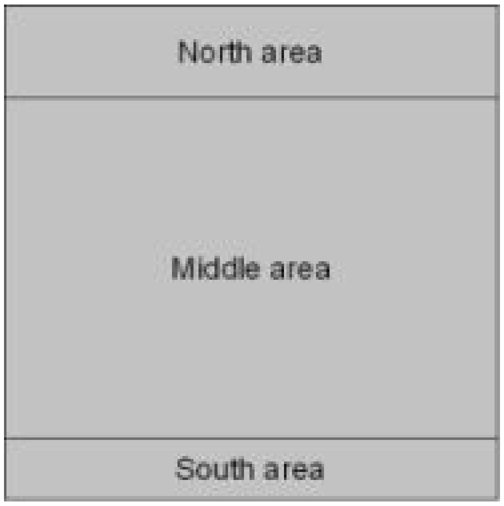


Figure 9.2. Page layout

The North area contains the application logo and title. The South area shows the copyright notice and has some utility web links. The Middle area consists of the specific menu component, followed by the content and sidebar sub-areas with the content on the left and the sidebar on the right (Figure 9.3). This is often called a 2 column layout [Page layouts]. Another popular format is a 3 column page layout.

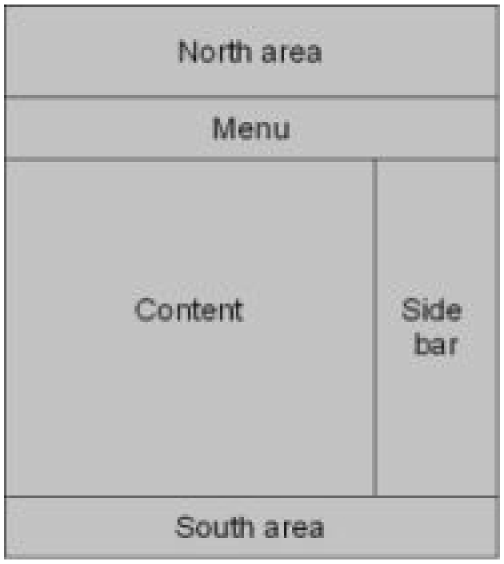


Figure 9.3. Page areas and sub-areas

The Content sub-area contains two sections: the graphical domain model and Category web links (Figure 9.4). The Side bar sub-area consists of three sections: Comments (C), Questions (Q) and Members in this spiral or Member interests (MI) in the next one. The choice of sections is purely pedagogical. The important point is to understand that a page is divided into sections and each section is presented by a web component. If a web component is available in a catalog of reusable components, the productivity of web page development will increase. Of course, a web component must be able to accept different parameters in a generic way, so that the same web component may be used with different domain models. If a web component is almost, but not quite suitable for user needs, a new web component should be developed, ideally based on an existing web component.

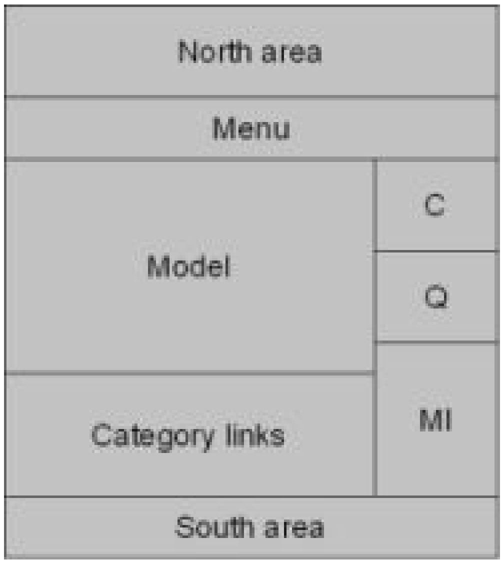
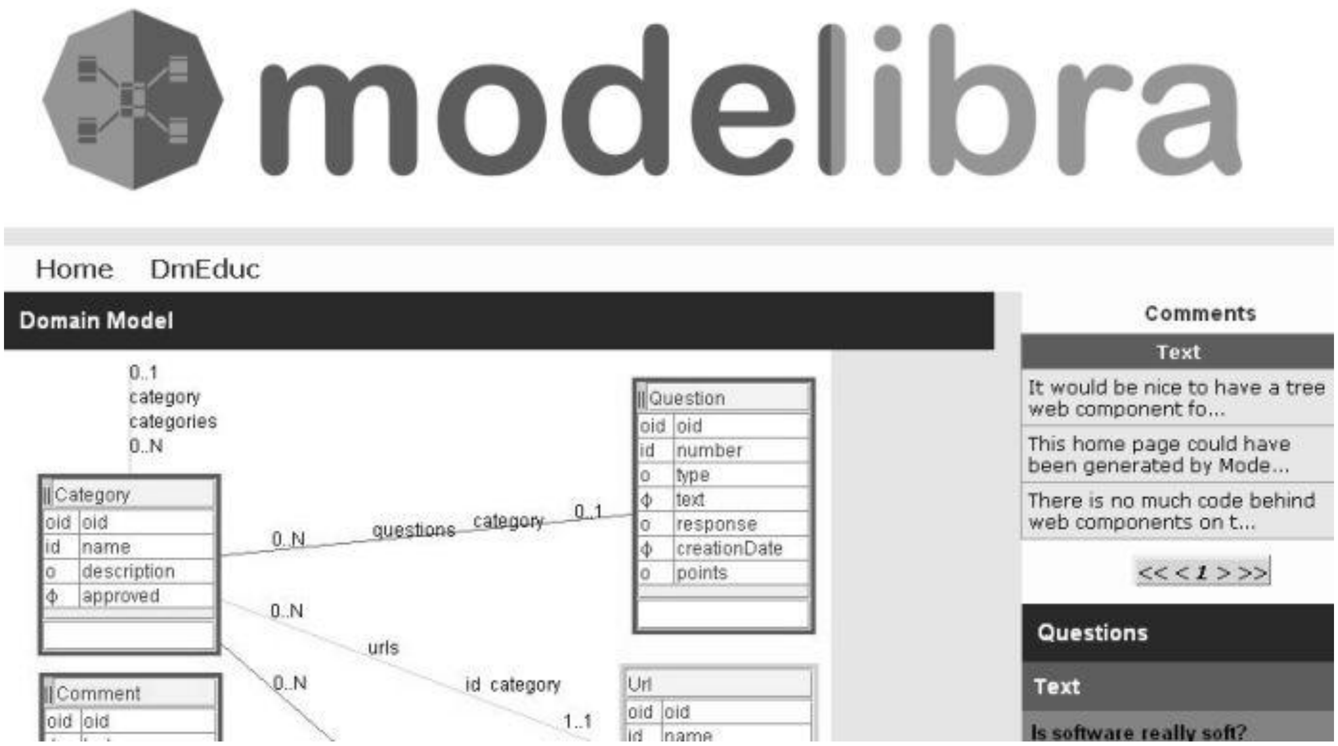


Figure 9.4. Page areas and sub-areas

In order to save some space, only the upper part of the home page is shown in Figure 9.5. The visible sections (some are completely visible and some only partially) are: the logo section, the menu section, the Comments section and the Questions section.



It would be nice to have a tree web component fo...

This home page could have been generated by Mode...

There is no much code behind web components on t...

Is software really soft?

Figure 9.5. Upper part of the home page

The logo section presents the Modelibra logo. The menu section provides two links, one to the Modelibra home page and the other to the DmEduc generic application. The generic application may be seen as the administration part of the complete web application. The model section presents the DmEduc domain model for this spiral. The Comments sections displays three existing comments. The Questions sections shows barely the first question. It is important to realize that the home page will change dynamically when a new comment is added in the DmEduc generic application.

Page HTML

The HomePage.html starts with a declaration that it is an XHTML document with the UTF-8 character encoding [UTF-8].

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
      "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
```

UTF-8 is an efficient way of encoding Unicode characters [Unicode]. Unicode provides a unique number for every character regardless of platform, software or natural language.

XHTML [XHTML] is HTML with some basic rules applied from XML [XML]. The rules are:

- XHTML elements are *properly nested*;
- XHTML elements are *closed*;
- XHTML elements are written in *lower case*;
- XHTML documents have only *one root element*.

The DOCTYPE declaration indicates which version of XHTML is used in the page. DOCTYPE is a key component of a compliant web pages and the page validation cannot be done without it [Web standards].

The wicket namespace [XML Namespaces] is declared to indicate that the XHTML elements will be extended with names that have the wicket prefix.

```
<html xmlns:wicket="http://wicket.apache.org/">
```

The head element provides information about the page document. It provides the page title. The meta element defines the author of the page. The link elements are used to reference CSS files that are located in the css directory of the web application. There are different CSS files. Each is made for a specific presentation purpose. For example, declarations for a menu can be found in the menu.css file. Not all CSS file are used in this page. However, since there is no performance penalty, it is easier to reference them all. If a page presentation is to be changed, all potential CSS declarations are already available.

There is also a link to a favorite icon called favicon. The favicon is an image file with 16 colors. The image is 16 pixel wide and 16 pixel high. A little icon shows up in a web browser in front of the URL.

```
<head>
      <title>DmEduc Home Page</title>
      <meta name="author" content="Dzenan Ridjanovic" />
      <link rel="stylesheet" type="text/css" media="screen" href="css/app.css" />
      <link rel="stylesheet" type="text/css" media="screen" href="css/box.css" />
      <link rel="stylesheet" type="text/css" media="screen" href="css/cite.css" />
      <link rel="stylesheet" type="text/css" media="screen" href="css/code.css" />
      <link rel="stylesheet" type="text/css" media="screen" href="css/def.css" />
```

```
	<link rel="stylesheet" type="text/css" media="screen" href="css/form.css" />
	<link rel="stylesheet" type="text/css" media="screen" href="css/layout.css" />
	<link rel="stylesheet" type="text/css" media="screen" href="css/layout2c.css"/
	>
	<link rel="stylesheet" type="text/css" media="screen"
		href="css/layout3c.css" />
	<link rel="stylesheet" type="text/css" media="screen" href="css/link.css" />
	<link rel="stylesheet" type="text/css" media="screen" href="css/list.css" />
	<link rel="stylesheet" type="text/css" media="screen" href="css/marker.css" />
	<link rel="stylesheet" type="text/css" media="screen" href="css/menu.css" />
	<link rel="stylesheet" type="text/css" media="screen" href="css/msg.css" />
	<link rel="stylesheet" type="text/css" media="screen" href="css/page.css" />
	<link rel="stylesheet" type="text/css" media="screen" href="css/section.css" /
	>
	<link rel="stylesheet" type="text/css" media="screen" href="css/slide.css" />
	<link rel="stylesheet" type="text/css" media="screen" href="css/table.css" />
	<link rel="shortcut icon" href="img/favicon.ico" />
</head>

<body>

...

</body>

</html>
```

The page areas, sub-areas and sections are defined in the body element by using div elements. In future, areas, sub-areas and sections will be called simply sections. The div element defines a division/section in a document. The class attribute refers to a specific CSS declaration in one of referenced .css files. The class attribute in CSS has nothing to do with a class concept in Java. The north, middle and south CSS classes come from the layout.css file located in the css directory of the project. The content and sidebar CSS classes are declared in the layout2.css file.

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

<div class="middle">
	<div wicket:id="homeMenu">
		To be replaced dynamically by the Home menu of links.
	</div>

	<div class="content">
		<div class="section-title">
			Domain Model
		</div>
		<div>
			<img src = "mm/DmEduc.jpg" alt="domain model" />
		</div>
		<div wicket:id="categoryNameUrlLinkList">
```

```
                To be replaced dynamically by the list of categories
                each with its web links.
            </div>
        </div>

        <div class="sidebar">
            <div wicket:id="commentTable">
                To be replaced dynamically by the table of comments.
            </div>
            <br/>
            <div wicket:id="questionTextList">
                To be replaced dynamically by the list of questions.
            </div>
            <br/>
            <div wicket:id="memberList">
                To be replaced dynamically by the list of members.
            </div>
        </div>
    </div>

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

The north section consists of an image of Modelibra. The south section contains the copyright text and links to various web page validation services.

The specific (developed in this spiral) menu component has a Wicket id (wicket:id) with the homeMenu name. The content of this web component will be determined dynamically by Wicket.

The content section starts with the Domain Model title followed by the model image. It ends with the generic (comes from ModelibraWicket) component whose Wicket id is categoryNameUrlLinkList.

The sidebar section has three web components that follow each other. The first two web components are generic (come from ModelibraWicket), while the last one is specific (developed in this spiral).

Page Java

The HomePage.java class extends the DmPage abstract class from ModelibraWicket, which in turn inherits its behavior from the WebPage class of Wicket.

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

import org.apache.wicket.markup.html.WebPage;
import org.modelibra.wicket.concept.EntityDisplayTablePanel;
import org.modelibra.wicket.concept.EntityPropertyDisplayListPanel;
```

```java
import org.modelibra.wicket.neighbor.ParentChildPropertyDisplayListPanel;
import org.modelibra.wicket.view.View;
import org.modelibra.wicket.view.ViewModel;

import dmeduc.DmEduc;
import dmeduc.weblink.WebLink;
import dmeduc.weblink.category.Categories;
import dmeduc.weblink.comment.Comments;
import dmeduc.weblink.member.Members;
import dmeduc.weblink.question.Questions;
import dmeduc.wicket.app.DmEducApp;
import dmeduc.wicket.weblink.member.MemberListPanel;

public class HomePage extends WebPage {

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

            // Menu
            add(new HomeMenu("homeMenu"));

            // Questions
            ViewModel questionsModel = new ViewModel();
            questionsModel.setModel(webLink);
            Questions questions = webLink.getQuestions();
            questionsModel.setEntities(questions);
            questionsModel.setPropertyCode("text");

            View questionsView = new View();
            questionsView.setWicketId("questionTextList");
            questionsView.setTitle("Questions");

            EntityPropertyDisplayListPanel questionTextList = new
                  EntityPropertyDisplayListPanel(
                        questionsModel, questionsView);
            add(questionTextList);

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

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

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

```
        // Category Urls
        ViewModel categoryUrlsModel = new ViewModel();
        categoryUrlsModel.setModel(webLink);
        Categories categories = webLink.getCategories();
        Categories orderedApprovedCategories = categories
                    .getApprovedCategories().getCategoriesOrderedByName();
        categoryUrlsModel.setEntities(orderedApprovedCategories);
        categoryUrlsModel.setPropertyCode("name");
        categoryUrlsModel.getUserProperties().addUserProperty("childNeighbor",
                    "urls");
        categoryUrlsModel.getUserProperties().addUserProperty("childProperty",
                    "link");

        View categoryUrlsView = new View();
        categoryUrlsView.setWicketId("categoryNameUrlLinkList");
        categoryUrlsView.setTitle("Category.WebLinks");

        ParentChildPropertyDisplayListPanel categoryNameUrlLinkList = new
             ParentChildPropertyDisplayListPanel(
                    categoryUrlsModel, categoryUrlsView);
        add(categoryNameUrlLinkList);

        // Members
        Members members = webLink.getMembers();
        Members orderedMembers = members.getMembersOrderedByLastFirstName(true);
        add(new MemberListPanel("memberList", orderedMembers.getList()));
    }

}
```

The home page has only a default constructor, which is the constructor without arguments. The first step done in the constructor is to get the WebLink model. The model is obtained from the DmEduc domain that comes from the DmEducApp web application. The getApplication method is inherited from Wicket.

```
    DmEducApp dmEducApp = (DmEducApp) getApplication();
    DmEduc dmEduc = dmEducApp.getDmEduc();
    WebLink webLink = dmEduc.getWebLink();
```

The specific menu component does not have any parameters other then the required Wicket id (it cannot be more specific) and it is constructed by the HomeMenu class that is located in the same package as the HomePage class. After the menu object is created it is added as a page component by the add method inherited from the Wicket's WebPage class.

```
    add(new HomeMenu("homeMenu"));
```

The same Wicket id is used in HomePage.html. This is the only non-standard element in the home page and it is there to allow Wicket to make a link between the HTML element and the web component.

```
    <div wicket:id="homeMenu">
         To be replaced dynamically by the Home menu of links.
```

```
</div>
```

The first (not necessarily first displayed; the display order is defined by the position of sections in the corresponding .html file) generic component used in the code of the HomePage class is located in the org.modelibra.wicket.concept package and has the EntityPropertyDisplayListPanel class name. The name is rather long but it tells us clearly that the component displays a list of values for only one property of an entity. The list comes from the entities used. The web component has two composite arguments. The first argument is an object of the ViewModel class. The second argument is an object of the View class. Both classes come from the org.modelibra.wicket.view package. The two composite arguments are constructed and their specific parameters are set. Then, the web component with those two filled arguments is constructed and added to the home page.

```
ViewModel questionsModel = new ViewModel();
questionsModel.setModel(webLink);
Questions questions = webLink.getQuestions();
questionsModel.setEntities(questions);
questionsModel.setPropertyCode("text");

View questionsView = new View();
questionsView.setWicketId("questionTextList");
questionsView.setTitle("Questions");

EntityPropertyDisplayListPanel questionTextList =
      new EntityPropertyDisplayListPanel(
            questionsModel, questionsView);
add(questionTextList);
```

The view model must know from what domain model and what entities the view is derived from. The entities are questions that are an entry into the webLink domain model. For this component the entity property name (code for Modelibra) must be given. The model, entities and propertyCode are predefined properties of the ViewModel class. The web component displays here a list of questions obtained from the text property.

The view must have a Wicket id, so that a web component may be related to an HTML section. The Wicket id is questionTextList.

```
<div wicket:id="questionTextList">
      To be replaced dynamically by the list of questions.
</div>
```

The view may have a title (key), which is here Questions. The real title text comes from the Questions key in the DmEducApp.properties file.

```
Questions=Questions
```

After the ViewModel and View arguments are defined, the web component may be constructed by using those two arguments.

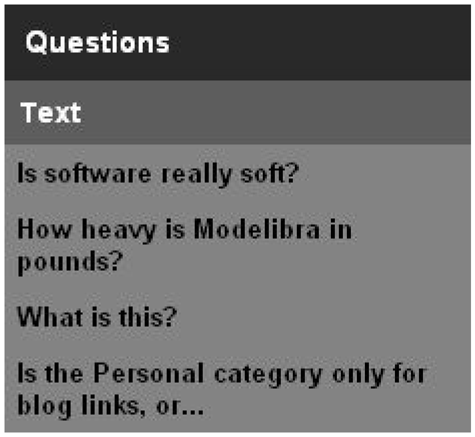
```
EntityPropertyDisplayListPanel questionTextList =
      new EntityPropertyDisplayListPanel(questionsModel, questionsView);
```

```
    add(questionTextList);
```

The Figure 9.6 presents visually the web component. The component has a title and a subtitle and a list of questions. The title key comes from the View argument. The subtitle comes from the DmEducApp.properties file based on the property code provided in the ViewModel argument. Modelibra finds the concept code from the given entities and, together with the property code, forms the Question.text key, which is then used to find the text to be displayed.

```
    Question.text=Text
```

If a question is long, only its beginning is displayed and the three dots are shown at the end. The default length is 48 characters.



Questions

Text

Is software really soft?

How heavy is Modelibra in pounds?

What is this?

Is the Personal category only for blog links, or...

Figure 9.6. Questions Web Component

This length may be changed by the domain's shortTextDefaultLength element in the XML configuration (see domain-config.dtd).

```
<!ELEMENT shortTextDefaultLength (CDATA)>
```

The Comments web component is similar to the Questions web component. It is defined on the single Comment concept based on the current entities of that concept. The display is of the table type and the concept's essential properties are displayed, here only the text property. The class of the web component is EntityDisplayTablePanel. Both the view model and the view of the web component are simple.

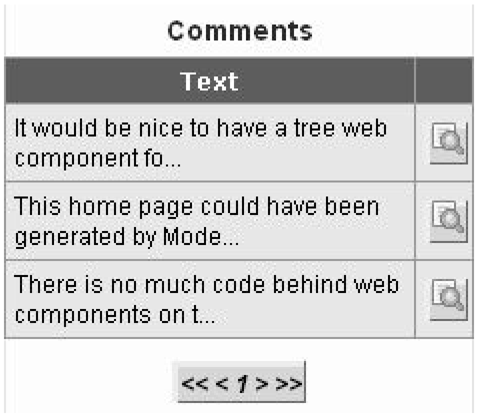
```
ViewModel commentsModel = new ViewModel();
commentsModel.setModel(webLink);
Comments comments = webLink.getComments();
commentsModel.setEntities(comments);

View commentsView = new View();
commentsView.setWicketId("commentTable");
commentsView.setContextView(homePageView);

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

```
                add(commentTable);
```

The view model sets the comments entities, while the view provides the commentTable Wicket id.

```
<div wicket:id="commentTable">
      To be replaced dynamically by the table of comments.
</div>
```

The Figure 9.7 presents a table of comments.



Comments

Text	
It would be nice to have a tree web component fo...	
This home page could have been generated by Mode...	
There is no much code behind web components on t...	

Figure 9.7. Comments Web Component

The title is derived by the web component from the provided entities with the help of the meta properties. The code of the entities (Comments) is used as the property key to find the property value (Comments) in the properties file. Note the difference between a text property used to find the display text and a concept property used to describe the concept).

```
Comments=Comments
```

The only property displayed is the only essential property defined as such in the reusable-domain-config.xml file.

```
<property oid="1171894920500">
      <code>text</code>
      <propertyClass>java.lang.String</propertyClass>
      <maxLength>1020</maxLength>
      <required>true</required>

      <essential>true</essential>
</property>
```

This definition has not been changed in the specific-domain-config.xml file.

```
<property oid="1171894920500">
      <code>text</code>

      <extension>true</extension>
      <extensionProperty>text</extensionProperty>
```

```
</property>
```

The property column title is derived by the web component from the provided entities with the help of the generic properties. The concept code and the property code are used together (Comment.text) as the property key to find the property value (Text) in the properties file.

```
Comment.text=Text
```

Each comment has a button-like link to display all details of the comment. This is useful if the comment's text is too long for the short text's default length. A click on the link will display a different web page. The context link on that page will display again the home page.

The Comments web component has a simple view model based on one concept. Often there is a need to display data based on the one-to-many relationship between two concepts. The ParentChildPropertyDisplayListPanel generic component displays one property from a parent entity and one property from a child entity but in a list of parents with the corresponding child list for each parent.

The parent-child web component is constructed based on its view model and view.

```
ViewModel categoryUrlsModel = new ViewModel();
categoryUrlsModel.setModel(webLink);
Categories categories = webLink.getCategories();
Categories orderedApprovedCategories = categories
           .getApprovedCategories().getCategoriesOrderedByName();
categoryUrlsModel.setEntities(orderedApprovedCategories);
categoryUrlsModel.setPropertyCode("name");
categoryUrlsModel.getUserProperties().addUserProperty(
           "childNeighbor", "urls");
categoryUrlsModel.getUserProperties().addUserProperty(
           "childProperty", "link");

View categoryUrlsView = new View();
categoryUrlsView.setWicketId("categoryNameUrlLinkList");
categoryUrlsView.setTitle("Category.WebLinks");

ParentChildPropertyDisplayListPanel categoryNameUrlLinkList =
      new ParentChildPropertyDisplayListPanel(
           categoryUrlsModel, categoryUrlsView);
add(categoryNameUrlLinkList);
```

In the view model, the categories entry point is obtained from the domain model and its subset of only approved categories, ordered by name, is set as entities. The property code is set to name. The ViewModel class has a generic way of adding additional, user oriented properties. Two user properties are defined here and they are used to find children of a parent and a child property that will be displayed. The childNeighbor user property is the urls neighbor of the Category concept. The childProperty user property is the link property of the Url concept.

The view determines the Wicket id for this component and its title. The Wicket id is categoryNameUrlLinkList.

```
<div wicket:id="categoryNameUrlLinkList">
     To be replaced dynamically by the list of categories
     each with its web links.
</div>
```

The title is the Category.WebLinks key from the DmEducApp_en.properties file. The specific properties file is used since this specific key does not exist in the generic properties file. The content of the generic properties file is generated. The content of the specific properties file is created by a developer.

```
Category.WebLinks=Category Web Links
```

The Figure 9.8 presents visually the web component. After the Category Web Links title, a list of categories is displayed. Each category has its title, which is a sub-title for the web component, and a list of web links. A web link in the list is clickable.

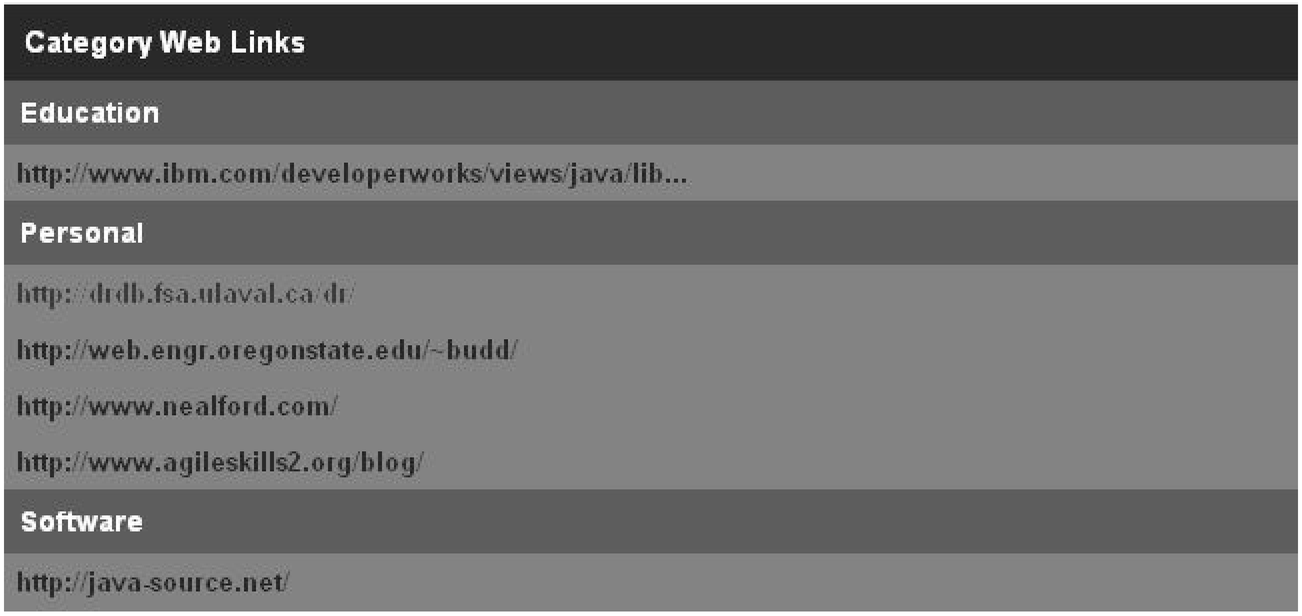

Category Web Links
Education
http://www.ibm.com/developerworks/views/java/lib...
Personal
http://drdb.fsa.ulaval.ca/dr/
http://web.engr.oregonstate.edu/~budd/
http://www.nealford.com/
http://www.agileskills2.org/blog/
Software
http://java-source.net/

Figure 9.8. Category Urls Web Component

Home Menu

In the HomePage class the menu component is constructed and added to the page.

```
add(new HomeMenu("homeMenu"));
```

The homeMenu Wicket id is used to link the menu component with the page section in the HomePage.html file.

```
<div wicket:id="homeMenu">
     To be replaced dynamically by the Home menu of links.
</div>
```

The HomeMenu class is relatively simple. It extends the DmPanel abstract class from ModelibraWicket, which in turn extends the Panel class from Wicket. Since the Panel class constructor requires the Wicket id as its only argument, the id is passed to the inheritance parent in the super keyword.

Note that the final keyword is used for the argument of the menu constructor. Wicket developers use the final keyword quite often and ModelibraWicket tries to be consistent with what the Wicket community does.

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

import org.apache.wicket.markup.html.link.Link;
import org.apache.wicket.markup.html.panel.Panel;
import org.modelibra.wicket.app.domain.DomainPage;

public class HomeMenu extends Panel {

	public HomeMenu(final String wicketId) {
		super(wicketId);
		Link domainLink = new Link("domainLink") {
			public void onClick() {
				setResponsePage(new DomainPage());
			}
		};
		add(domainLink);
	}

}
```

There are two links in the home page menu. Figure 9.9 shows the menu section.

Figure 9.9. Home Page Menu Section

The two links can easily be recognized in the HomeMenu.html file. This HTML file is different from an HTML file for a web page. It contains only a definition of elements used in the panel (or the page section). Thanks to Wicket, an object oriented approach to defining even HTML elements of a web component can be practiced. Wicket requires the use of the wicket name space and the wicket:panel element. Since the panel is a part of a web page, it is Wicket that will insert the HTML code with dynamic data into the section of the web page where the Wicket id is used to make a link between the page section and the web component.

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

<html xmlns:wicket>

<wicket:panel>

<div class="menu">
```

```
	<ul>
		<li>
			<a href="http://www.modelibra.org/">Home</a>
		</li>
		<li>
			<a wicket:id="domainLink">DmEduc</a>
		</li>
	</ul>
</div>

</wicket:panel>

</html>
```

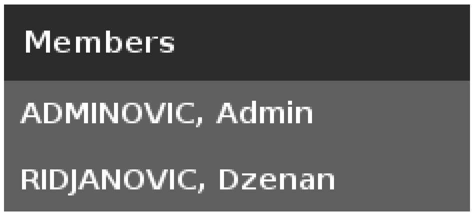
The first link is static (actually its value is static; it does not change dynamically) and points to the home page of the Modelibra software family. The second link is dynamic (actually its value is constructed dynamically) and is created by Wicket based on the Wicket id. This creation happens in the HomeMenu class.

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

The link is constructed and added to the menu. Since the Link class from Wicket is abstract, an anonymous class [Anonymous class] is defined that provides a reaction to the on-click event. An anonymous class is a local (or inner) class without a name. This reaction is a display of the domain page that is the principal page of the generic web application of ModelibraWicket. Note that the domain page is constructed only if the on click event happens.

Specific Component

The ParentChildPropertyDisplayListPanel generic component displays one property from a parent entity and one property from a child entity but in a list of parents with the corresponding child list for each parent. This generic component is not very flexible since it displays only one property from both a parent and a child. Ideally, we would like to have a more generic component or a different generic component that will fulfill the requirement of displaying more than one property. However, it is always possible to make our own specific component with that requirement. The MemberListPanel component in Figure 9.10 displays two properties from the Member concept.


Members

ADMINOVIC, Admin

RIDJANOVIC, Dzenan

Figure 9.10. Members Section

After the *Members* title, a list of members is displayed. Each member is presented with his last name in capital letters, comma and then his first name.

The specific web component consists of one Java class and one HTML file, both with the same name but different file extension. The name of the component is MemberListPanel. The name indicates that the component is a panel that will be used for a section of a web page. The members are displayed in a list. The class of the web component is located in the dmeduc.wicket.weblink.member package. The HTML file of the web component is placed in the same directory.

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

import java.util.List;

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

import dmeduc.weblink.member.Member;

public class MemberListPanel extends Panel {

	public MemberListPanel(final String wicketId, final List<Member> memberList) {
		super(wicketId);
		add(new MemberList("memberList", memberList));
	}

	/**
	 * Member list view as an inner class.
	 */
	private class MemberList extends ListView {

		public MemberList(final String wicketId, final List<Member> memberList) {
			super(wicketId, memberList);
		}

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

```
    }

}
```

The web component extends the Panel class from Wicket. Its constructor has two arguments. The first argument is a Wicket id that is required by the Panel class. The second argument is a list of members. There is one private inner class [Inner] called MemberList that extends ListView from Wicket. An object of the inner class is added to the panel. The inner class has the same arguments as the specific web component. The ListView class requires implementation of the protected method called populateItem. The ListView class is an abstract class that has the populateItem abstract method. This method, when implemented in a specific class, will be called by Wicket to populate a list of objects. In our case it is a list of entities. More precisely it is a list of members.

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

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

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

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

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

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

and not

```
    add(memberNameLabel);
```

Note that Wicket will call the populateItem method as many times as there are members. It is also Wicket that will use the corresponding HTML code to display a list of members.

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

<html xmlns:wicket>

<wicket:panel>
```

```
    <div class="section-title">
        Members
    </div>

    <div wicket:id = "memberList">
        <div wicket:id = "memberName" class = "box-title">
            Member name
        </div>
    </div>

</wicket:panel>

</html>
```

The MemberListPanel HTML definition can be considered as the section divided into two subsections, one for the specific title, displayed according to the section-title CSS class, and the other for the list of member names.

Summary

The specific home page is created by reusing generic components of ModelibraWicket. What was not possible to do with generic components is done by creating a specific web component. The specific web component is developed based on Wicket components. The home page is designed by decomposing it into sections. A section is presented through a web component. A web component may be specific or generic. It is always easier to reuse a generic web component than to develop a specific web component. However, if there is no generic web component that satisfies requirements of the page section, a specific component must be developed. A generic component is developed by an advanced web programmer to support the needs of different web applications, and in such a way increase the productivity of application programmers.

The first specific web component is developed from scratch. It displays a list of member names, where a name shows the last name in capital letters followed by the first name.

A generic web component uses a part of the domain model, called a view model. ModelibraWicket uses the Modelibra domain model framework for its view models, and the Wicket web framework for its web views. The Wicket web framework is used extensively to develop specific and generic web components of ModelibraWicket. Wicket is one of rare web frameworks that concentrates the effort of developing web components in Java classes. Of course, the knowledge of HTML and CSS is essential, but programming is done only in Java classes. The HTML and CSS technologies are used only to decide how web components are presented to users of the web application.

The next chapter will show how standard features of a web application are developed by ModelibraWicket and Wicket. It will also explain how to add member interests to the specific web component.

Questions

1. How many different technologies are used to create a web component?
2. Why a view model is used in a web component?
3. Why a Wicket id is called an identifier?
4. What is a difference between a web component and a Java class?

Exercises

Exercise 9.1.

Create an About page and move the graphical model from the home page to the new page.

Exercise 9.2.

Use a new generic web component from the org.modelibra.wicket.concept package in ModelibraWicket.

Exercise 9.3.

Develop a new specific web component that will be based on the one-to-many relationship between two concepts and that will display two properties from a parent and two properties from a child.

Web Links

[Anonymous class] Anonymous class
http://www.developer.com/java/other/article.php/3300881

[DOCTYPE] DOCTYPE
http://www.alistapart.com/stories/doctype/

[final keyword] The Final Word On the final Keyword
http://renaud.waldura.com/doc/java/final-keyword.shtml

[Inner] Inner class
https://java.sun.com/docs/books/tutorial/java/javaOO/innerclasses.html

[Page layouts] Sample CSS Page Layouts
http://www.maxdesign.com.au/presentation/page_layouts/

[Unicode] Unicode
http://unicode.org/

[UTF-8] UTF-8 character encoding
http://www.utf-8.com/

[Web standards] Web standards
http://www.w3.org/QA/2002/04/Web-Quality

[XHTML] XHTML
http://www.w3.org/TR/xhtml1/

[XML] XML
http://www.w3.org/TR/REC-xml/

[XML Namespaces] XML Namespaces
http://www.w3schools.com/xml/xml_namespaces.asp