Chapter 13: Ajax Components

The objective of this chapter is to introduce, in the new application called Course Presentation, a few web components developed in Ajax [Ajax]. With the Course Presentation application you can create web slide presentations. A slide has points, where each point is a short text, a formatted explanation text, a web link, an image or a programming code. Images may be uploaded to a server to be used in image point definitions. A presentation may be shown in a slide display with the first, next, prior and last navigation links. There is also a continuous sideshow.

Domain Model

The domain of the Course Presentation web application is Course and its principal model is called Lecture.

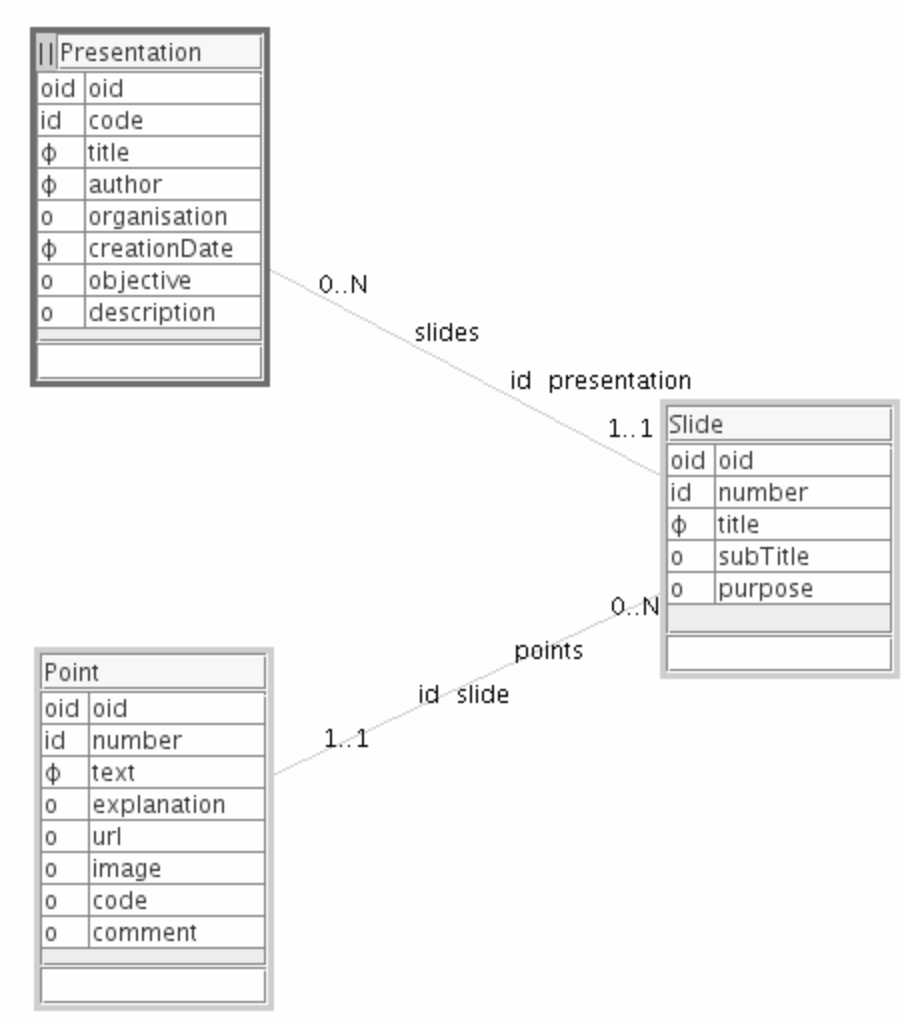

Figure 13.1. Course Lecture

The model contains three concepts: Presentation, Slide and Point. The Presentation concept is an entry

point into the model. A presentation may have many slides. A slide may have several points. In addition to oid, a point must have a sequential number and a short text. A number is auto incremented by 1, starting with 0. This is defined in the XML reusable configuration.

```
<property oid="1176413579105">
      <code>number</code>
      <propertyClass>
            java.lang.Integer
      </propertyClass>
      <required>true</required>
      <autoIncrement>true</autoIncrement>
      <unique>true</unique>

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

In the XML specific configuration the number property is declared as not updateable by a user.

```
<property oid="1176413579105">
      <code>number</code>
      <extension>true</extension>
      <extensionProperty>
            number
      </extensionProperty>

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

The text property is required and with the maximum length of 255 characters. The entered text cannot be formatted. If there is a need for a formatted text, the longer (1020 characters) explanation property may be used. A point may also be a web link (url), an uploaded image, or a programming code. In those cases, the text property is not displayed. The length of a programming code may be up to 4080. A comment may be added to provide some additional information about the point.

There is also the Reference model with the Member, SecurityRole and CountryLanguage concepts.

Home Page

The home page

```
package course.wicket.app.home;

import org.apache.wicket.markup.html.panel.FeedbackPanel;
import org.apache.wicket.markup.html.panel.Panel;
import org.modelibra.wicket.concept.EntityDisplayTablePanel;
import org.modelibra.wicket.concept.EntityPropertyDisplayListPanel;
import org.modelibra.wicket.container.DmPage;
import org.modelibra.wicket.neighbor.ParentChildPropertyDisplayListPanel;
import org.modelibra.wicket.security.AppSession;
```

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

import course.Course;
import course.lecture.Lecture;
import course.lecture.presentation.Presentations;
import course.reference.Reference;
import course.reference.member.Member;
import course.reference.member.Members;
import course.wicket.app.CourseApp;

public class HomePage extends DmPage {

	public HomePage() {
		...
	}

}
```

displays a table of presentations, from which the selected presentation may be shown slide by slide by following the next link.

```
		CourseApp courseApp = (CourseApp) getApplication();
		Course course = courseApp.getCourse();
		Lecture lecture = course.getLecture();

		...

		// Presentations
		ViewModel presentationsViewModel = new ViewModel();
		presentationsViewModel.setModel(lecture);
		Presentations presentations = lecture.getPresentations();
		Presentations presentationsOrderedByCode = presentations
				.getPresentationsOrderedByCode(true);
		presentationsViewModel.setEntities(presentationsOrderedByCode);

		View presentationsView = new View();
		presentationsView.setWicketId("presentationTableSection");
		presentationsView.setPage(this);

		EntityDisplayTablePanel presentationTableSection = new
	EntityDisplayTablePanel(
				presentationsViewModel, presentationsView);
		add(presentationTableSection);
```

In the sidebar of the home page, for each presentation, a list of slide headers are displayed.

```
		// Presentation Slides
		ViewModel presentationSlidesViewModel = new ViewModel();
		presentationSlidesViewModel.setModel(lecture);
```

```
            Presentations presentationsOrderedByTitle = presentations
                        .getPresentationsOrderedByTitle(true);
            presentationSlidesViewModel
                        .setEntities(presentationsOrderedByTitle);
            presentationSlidesViewModel.setPropertyCode("title");
            presentationSlidesViewModel.getUserProperties().addUserProperty(
                        "childNeighbor", "slides");
            presentationSlidesViewModel.getUserProperties().addUserProperty(
                        "childProperty", "header");

            View presentationSlidesView = new View();
            presentationSlidesView.setWicketId("presentationSlideListSection");
            presentationSlidesView.setTitle("Presentation.Slides");

            ParentChildPropertyDisplayListPanel presentationSlideListSection =
new ParentChildPropertyDisplayListPanel(
                        presentationSlidesViewModel, presentationSlidesView);
            add(presentationSlideListSection);
```

The slide header is a derived property of the Slide concept. A derived property is declared in the specific XML configuration.

```
<property oid="1176413611970">
      <code>header</code>
      <propertyClass>
            java.lang.String
      </propertyClass>
      <derived>true</derived>

      <essential>false</essential>
</property>
```

The derived method is provided in the specific Slide class. It derives a header from the slide title and subTitle regular properties.

```
public String getHeader() {
      String header = getTitle();
      if (getSubTitle() != null) {
            header = header + ": " + getSubTitle();
      }
      return header;
}
```

The corresponding HTML code for the two sections is:

```
<div class="content">
      <div wicket:id="presentationTableSection">
            To be replaced dynamically by the table of presentations.
      </div>
</div>
```

```
<div class="sidebar">
      ...
      <div wicket:id="presentationSlideListSection">
            To be replaced dynamically by the list of presentation titles
            each with its slide titles.
      </div>
</div>
```

Slideshow

After a presentation is selected, by the *Slides* link, in the table of presentations, the presentation's slides may be traversed by clicking on the next link (>). or a slideshow may be started by clicking on the play link, which is found to the left of the drop down choice of slide duration times in seconds (Figure 13.2). If the duration of 30 seconds is chosen, each slide will be displayed for 30 seconds and the presentation will move to the next slide. If there is no the next slide, the presentation will turn around and restart from the first slide. The number of minutes passed from the beginning of the presentation is shown as well.

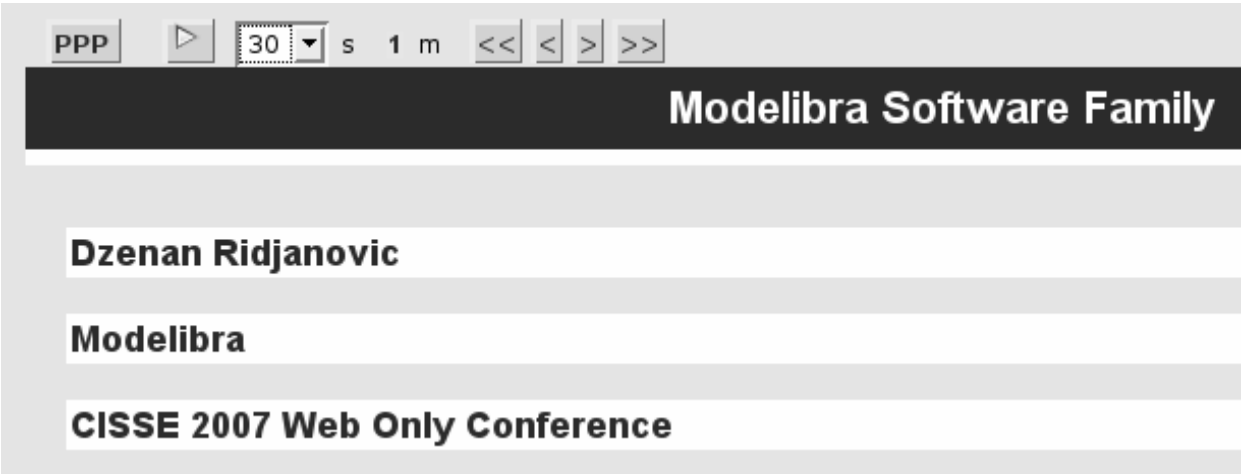

Figure 13.2. Slideshow navigation bar

In the XML specific configuration the display type of the Slide concept is declared as slide.

```
<displayType>slide</displayType>
```

Thus, ModelibraWicket uses the EntityDisplaySlidePage web component. Since the specific version of the component exists in the course.wicket.lecture.slide specific package, the display of slides is determined by the following code.

```
package course.wicket.lecture.slide;

import org.apache.wicket.markup.html.WebPage;
import org.apache.wicket.markup.html.panel.Panel;
import org.modelibra.wicket.concept.navigation.AjaxEntitySlideshowPanel;
import org.modelibra.wicket.concept.navigation.AjaxFallbackEntityNavigatePanel;
```

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

import course.lecture.slide.Slide;
import course.lecture.slide.Slides;

public class EntityDisplaySlidePage extends WebPage {

	public EntityDisplaySlidePage(ViewModel viewModel, View view) {
		add(homePageLink("homePage"));

		ViewModel slideViewModel = new ViewModel();
		slideViewModel.copyPropertiesFrom(viewModel);

		View slideView = new View();
		slideView.copyPropertiesFrom(view);
		slideView.setWicketId("slideDisplaySlidePanel");

		Slides slides = (Slides) viewModel.getEntities();
		Slide slide = (Slide) viewModel.getEntity();
		Slide firstSlide = (Slide) slides.first();
		Panel slideDisplaySlidePanel;
		if (firstSlide == null) {
			slideDisplaySlidePanel = new Panel("slideDisplaySlidePanel");
			slideDisplaySlidePanel.setVisible(false);
		} else {
			if (slide == null) {
				Slides orderedSlides = slides.getSlidesOrderedByNumber(true);
				slideViewModel.setEntities(orderedSlides);
				slideViewModel.setEntity(firstSlide);
			} else {
				slideViewModel.setEntities(slides);
				slideViewModel.setEntity(slide);
			}
			slideDisplaySlidePanel = new SlideDisplaySlidePanel(slideViewModel,
					slideView);
			slideDisplaySlidePanel.setOutputMarkupId(true);
		}
		add(slideDisplaySlidePanel);

		// Ajax slideshow.
		ViewModel slideshowViewModel = new ViewModel();
		slideshowViewModel.copyPropertiesFrom(slideViewModel);

		View slideshowView = new View();
		slideshowView.copyPropertiesFrom(slideView);
		slideshowView.setWicketId("slideshow");

		AjaxEntitySlideshowPanel ajaxEntitySlideshowPanel = new
			AjaxEntitySlideshowPanel(slideshowViewModel, slideshowView) {
```

```
            @Override
            protected String getNavigatedPanelId() {
                  return "slideDisplaySlidePanel";
            }

            @Override
            protected Panel getNavigatedPanel(final ViewModel viewModel,
                        final View view) {
                  return new SlideDisplaySlidePanel(viewModel, view);
            }
      };
      add(ajaxEntitySlideshowPanel);

      // Ajax minute counter.
      add(new AjaxMinuteCounterPanel("minuteCounter"));

      // Ajax slide navigator uses the same ViewModel as
      //   AjaxEntitySlideshowPanel.

      View navigateView = new View();
      navigateView.copyPropertiesFrom(slideshowView);
      navigateView.setWicketId("navigator");

      AjaxFallbackEntityNavigatePanel ajaxEntitySlideNavigatePanel = new
            AjaxFallbackEntityNavigatePanel(slideshowViewModel, navigateView) {

            @Override
            protected String getNavigatedPanelId() {
                  return "slideDisplaySlidePanel";
            }

            @Override
            protected Panel getNavigatedPanel(final ViewModel viewModel,
                        final View view) {
                  return new SlideDisplaySlidePanel(viewModel, view);
            }
      };
      add(ajaxEntitySlideNavigatePanel);
   }

}
```

The EntityDisplaySlidePage specific web page is decomposed in four sections: a slide display section with slide header and points, a slideshow section with a link to start a presentation and a drop down choice to select a time interval for turning slides, a minute counter section to allow a presenter to see how many minutes have passed from the beginning of the presentation, and a slide navigator with the first, next, prior and last links.

The EntityDisplaySlidePage page has the following HTML code.

```
<body>
```

```
       
      <a class="button" wicket:id="homePage">
          PPP
      </a>
       
       
      <span wicket:id="slideshow"></span>
       
      <span wicket:id="minuteCounter"/>
       
      <span wicket:id="navigator"/>
      <br/>
      <div wicket:id="slideDisplaySlidePanel"/>
</body>
```

A slide is displayed by the specific SlideDisplaySlidePanel component from the same package. A slide has a title, a subtitle and a list of points.

```
package course.wicket.lecture.slide;

import org.apache.wicket.markup.html.basic.Label;
import org.apache.wicket.markup.html.panel.Panel;
import org.apache.wicket.model.PropertyModel;
import org.modelibra.wicket.view.View;
import org.modelibra.wicket.view.ViewModel;

import course.lecture.slide.Slide;
import course.wicket.lecture.point.PointDisplayTableListView;

public class SlideDisplaySlidePanel extends Panel {

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

            Slide slide = (Slide) viewModel.getEntity();
            add(new Label("title", new PropertyModel(slide, "title")));
            add(new Label("subTitle", new PropertyModel(slide, "subTitle")));

            add(new PointDisplayTableListView(
                        "pointDisplayTableListView", slide.getPoints()));
      }

}
```

The corresponding HTML code of the specific slide component is:

```
<wicket:panel>

<div>
      <div class="slide-title" wicket:id="title">
            Slide title will be displayed here.
```

```
	</div>
	<div class="slide-subtitle" wicket:id="subTitle">
		Slide subtitle will be displayed here.
	</div>
	<br/>
	<ul>
		<li wicket:id="pointDisplayTableListView">
			<div class="slide-point" wicket:id="propertyValue">
				Point text or url will be displayed here.
			</div>
			<br/>
		</li>
	</ul>
</div>

</wicket:panel>
```

A list of points is displayed by the specific PointDisplayTableListView class.

```
package course.wicket.lecture.point;

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 org.modelibra.wicket.view.View;
import org.modelibra.wicket.view.ViewModel;
import org.modelibra.wicket.widget.CodeMultiLineLabelPanel;
import org.modelibra.wicket.widget.ContextImagePanel;
import org.modelibra.wicket.widget.ExternalLinkPanel;
import org.modelibra.wicket.widget.LabelPanel;
import org.modelibra.wicket.widget.MultiLineLabelPanel;

import course.Course;
import course.lecture.Lecture;
import course.lecture.point.Point;
import course.lecture.point.Points;
import course.wicket.app.CourseApp;

public class PointDisplayTableListView extends ListView {

	private CourseApp presentApp;

	private Course course;

	private Lecture lecture;

	public PointDisplayTableListView(String id, Points points) {
		super(id, points.getList());
		presentApp = (CourseApp) getApplication();
		course = presentApp.getCourse();
		lecture = course.getLecture();
	}
```

```
	protected void populateItem(ListItem item) {
		Point point = (Point) item.getModelObject();
		Panel pointPanel;
		if (point.getExplanation() != null) {
			ViewModel pointViewModel = new ViewModel();
			pointViewModel.setModel(lecture);
			pointViewModel.setEntity(point);
			pointViewModel.setPropertyCode("explanation");
			View pointView = new View();
			pointView.setWicketId("propertyValue");
			pointPanel = new MultiLineLabelPanel(pointViewModel, pointView);
		} else if (point.getUrl() != null) {
			ViewModel urlViewModel = new ViewModel();
			urlViewModel.setModel(lecture);
			urlViewModel.setEntity(point);
			urlViewModel.setPropertyCode("url");
			View urlView = new View();
			urlView.setWicketId("propertyValue");
			urlView.getUserProperties().addUserProperty("displayText",
					point.getText());
			pointPanel = new ExternalLinkPanel(urlViewModel, urlView);
		} else if (point.getImage() != null) {
			ViewModel imageViewModel = new ViewModel();
			imageViewModel.setModel(lecture);
			imageViewModel.setEntity(point);
			imageViewModel.setPropertyCode("image");
			View imageView = new View();
			imageView.setWicketId("propertyValue");
			pointPanel = new ContextImagePanel(imageViewModel, imageView);
		} else if (point.getCode() != null) {
			ViewModel codeViewModel = new ViewModel();
			codeViewModel.setModel(lecture);
			codeViewModel.setEntity(point);
			codeViewModel.setPropertyCode("code");
			View codeView = new View();
			codeView.setWicketId("propertyValue");
			pointPanel = new CodeMultiLineLabelPanel(codeViewModel, codeView);
		} else {
			ViewModel pointViewModel = new ViewModel();
			pointViewModel.setModel(lecture);
			pointViewModel.setEntity(point);
			pointViewModel.setPropertyCode("text");
			View pointView = new View();
			pointView.setWicketId("propertyValue");
			pointPanel = new LabelPanel(pointViewModel, pointView);
		}
		item.add(pointPanel);
	}

}
```

If a point has an explanation, the MultiLineLabelPanel widget from ModelibraWicket is used. This widget allows formatting of lines and paragraphs. If a point does not have an explanation and has a url, the ExternalLinkPanel widget is used. If a point has an image, the ContextImagePanel widget is used. This image must be first uploaded. If a point has a code, the CodeMultiLineLabelPanel widget is used to display the programming code in a standard way. Finally, if all previous properties do not have a value, the text property is displayed with the LabelPanel widget.

The specific EntityDisplaySlidePage displays a slide together with three components done in Ajax [Ajax]. Ajax is an abbreviation for **A**synchronous **Ja**vaScript and **X**ML It is not a new technology, but a combination of old technologies used for creating highly interactive and responsive Web pages. A small amount of data is exchanged with a server in such a way to update only a page section. JavaScript is the scripting language in which function calls are made. Data exchange is carried by the XMLHttpRequest object.

In order to understand the basics of Ajax within the Wicket context, the AjaxMinuteCounterPanel utility class from ModelibraWicket will be explained first. The widget may be used to show the number of minutes passed from time 0, which is determined by the construction of the widget. The widget is a panel with one label. The label represents a minute counter whose Wicket model is implemented as the MinuteCounterModel private class. The minute counter has the Ajax subcomponent that updates the label every minute.

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

import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;
import org.apache.wicket.ajax.AjaxSelfUpdatingTimerBehavior;
import org.apache.wicket.markup.html.basic.Label;
import org.apache.wicket.markup.html.panel.Panel;
import org.apache.wicket.model.AbstractReadOnlyModel;
import org.apache.wicket.util.time.Duration;
import org.modelibra.util.Transformer;

public class AjaxMinuteCounterPanel extends Panel {

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

	private Label minuteCounter;

	public AjaxMinuteCounterPanel(String wicketId) {
		super(wicketId);
		minuteCounter = new Label("minuteCounter", new MinuteCounterModel());
		minuteCounter.add(new
			AjaxSelfUpdatingTimerBehavior(Duration.seconds(60)));
		add(minuteCounter);
	}

	private class MinuteCounterModel extends AbstractReadOnlyModel {

		private double startTime;
```

```
            public MinuteCounterModel() {
                  startTime = System.currentTimeMillis();
            }

            public Object getObject() {
                  double currentTime = System.currentTimeMillis();
                  double timeIntervalInSeconds = (currentTime - startTime) / 1000.0;
                  double timeIntervalInMinutes = timeIntervalInSeconds / 60.0;
                  Double timeIntervalInMinutesDouble = Transformer
                              .doubleDecimal(timeIntervalInMinutes);
                  int timeIntervalInMinutesInt = timeIntervalInMinutesDouble
                              .intValue();
                  return Transformer.string(timeIntervalInMinutesInt);
            }

      }

}
```

The Wicket model of the minute counter is read only. It implements the abstract getObject method. The method returns the difference between the start time created in the constructor and the current time. The difference is in minutes expressed as the String value.

The minute counter is expressed in HTML as the Wicket panel element.

```
<wicket:panel>

    <span wicket:id = "minuteCounter" class="marker">
        Minute counter
    </span> m

    <wicket:child/>

</wicket:panel>
```

The name of the generic web component, which is used to turn a slideshow, is AjaxEntitySlideshowPanel. It is an abstract class that requires the implementation of two methods: getNavigatedPanelId and getNavigatedPanel. In the case of EntityDisplaySlidePage the navigated panel id is slideDisplaySlidePanel and the navigated panel is SlideDisplaySlidePanel with the usual view model and view arguments.

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

import java.util.Arrays;
import java.util.List;

import org.apache.wicket.ajax.AbstractAjaxTimerBehavior;
import org.apache.wicket.ajax.AjaxRequestTarget;
import org.apache.wicket.ajax.form.AjaxFormComponentUpdatingBehavior;
import org.apache.wicket.ajax.markup.html.AjaxLink;
import org.apache.wicket.markup.html.form.DropDownChoice;
```

```
import org.apache.wicket.markup.html.image.Image;
import org.apache.wicket.markup.html.panel.Panel;
import org.apache.wicket.model.Model;
import org.apache.wicket.util.time.Duration;
import org.modelibra.IEntities;
import org.modelibra.IEntity;
import org.modelibra.wicket.container.DmPanel;
import org.modelibra.wicket.view.View;
import org.modelibra.wicket.view.ViewModel;

public abstract class AjaxEntitySlideshowPanel extends DmPanel {

	private static final Integer DEFAULT_UPDATE_INTERVAL = 5;

	private static final List<Integer> UPDATE_INTERVAL_LIST = Arrays.asList(5,
			10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60);

	private ViewModel viewModel;

	private View view;

	private boolean play = false;

	private SlideTimer slideTimer;

	private AjaxLink playLink;

	private AjaxLink stopLink;

	public AjaxEntitySlideshowPanel(final ViewModel viewModel, final View view) {
		super(view.getWicketId());
		this.viewModel = viewModel;
		this.view = view;

		slideTimer = new SlideTimer(Duration.seconds(DEFAULT_UPDATE_INTERVAL));
		add(slideTimer);

		Integer slideTimerUpdateInterval = DEFAULT_UPDATE_INTERVAL;
		DropDownChoice timerUpdateIntervalChoice = new DropDownChoice(
				"timerUpdateIntervalChoice",
				new Model(slideTimerUpdateInterval), UPDATE_INTERVAL_LIST);
		timerUpdateIntervalChoice.add(new AjaxFormComponentUpdatingBehavior(
				"onchange") {
			@Override
			protected void onUpdate(final AjaxRequestTarget target) {
				slideTimer.changeUpdateInterval((Integer) getComponent()
						.getModelObject());
			}
		});
		add(timerUpdateIntervalChoice);

		// Ajax link used to play a slideshow.
```

```
        playLink = new AjaxLink("playLink") {
            @Override
            public void onClick(final AjaxRequestTarget target) {
                play = true;
                stopLink.setVisible(true);
                playLink.setVisible(false);
                target.addComponent(playLink);
                target.addComponent(stopLink);
            }
        };
        playLink.add(new Image("play"));
        add(playLink.setOutputMarkupId(true)
                .setOutputMarkupPlaceholderTag(true));

        // Ajax link used to stop (actually ignore) the slide timer.
        stopLink = new AjaxLink("stopLink") {
            @Override
            public void onClick(final AjaxRequestTarget target) {
                play = false;
                stopLink.setVisible(false);
                playLink.setVisible(true);
                target.addComponent(playLink);
                target.addComponent(stopLink);
            }
        };
        stopLink.add(new Image("stop"));
        add(stopLink.setOutputMarkupId(true)
                .setOutputMarkupPlaceholderTag(true).setVisible(false));
    }

    private void setCurrentEntity(IEntity<?> currentEntity) {
        viewModel.setEntity(currentEntity);
    }

    private IEntity<?> getNext() {
        IEntities entities = viewModel.getEntities();
        IEntity<?> currentEntity = viewModel.getEntity();
        IEntity<?> nextEntity = entities.next(currentEntity);
        if (nextEntity == null) {
            nextEntity = entities.first();
        }
        return nextEntity;
    }

    private void replacePanel(final AjaxRequestTarget target) {
        Panel currentNavigatedPanel = (Panel) getParent().get(
                getNavigatedPanelId());

        view.setWicketId(currentNavigatedPanel.getId());

        Panel newNavigatedPanel = getNavigatedPanel(viewModel, view);
        newNavigatedPanel.setOutputMarkupId(true);
```

```
            currentNavigatedPanel.replaceWith(newNavigatedPanel);
            target.addComponent(newNavigatedPanel);
      }

      private class SlideTimer extends AbstractAjaxTimerBehavior {

            private SlideTimer(final Duration updateInterval) {
                  super(updateInterval);
            }

            @Override
            protected void onTimer(final AjaxRequestTarget target) {
                  if (play) {
                        setCurrentEntity(getNext());
                        replacePanel(target);
                  }
            }

            private void changeUpdateInterval(Integer updateInterval) {
                  setUpdateInterval(Duration.seconds(updateInterval));
            }

      }

      protected abstract String getNavigatedPanelId();

      protected abstract Panel getNavigatedPanel(final ViewModel viewModel,
                  final View view);

}
```

The AjaxEntitySlideshowPanel generic component is a panel. Its logic is based on the current entity obtained from the view model argument. In our case, it is a slide entity. It has a private inner class that defines a slide timer. The SlideTimer class extends the AbstractAjaxTimerBehavior class. It implements the onTimer method. If the slideshow is on, the method sets the current entity to the next one and replaces the current slide with the next slide. The private replacePanel method gets the current navigated panel based on the panel id. It then prepares the arguments for the getNavigatedPanel method to obtain the new navigated panel in order to replace the current panel with the new one. The new panel is added to the Ajax target from the onTimer method.

With the drop down choice the timer interval may be changed. An object of the AjaxFormComponentUpdatingBehavior class is added to the the drop down choice component to change the timer interval in the onUpdate method with the value chosen by a user.

There are two Ajax links to start (play) and stop the presentation. Both links are added to the Ajax target argument of the onClick method. However, only one of them is visible at the same time. The play link is visible before the presentation is started by a user.

The slideshow component has the corresponding HTML code with two links and a drop down choice.

```
<wicket:panel>
	<a class = "button" wicket:id="playLink">
		<img wicket:id = "play" src = "media-playback-start.png"
			width="16" height="16" />
	</a>
	<a class = "button" wicket:id="stopLink">
		<img wicket:id = "stop" src = "media-playback-stop.png"
			width="16" height="16" />
	</a>
	 
	<select wicket:id="timerUpdateIntervalChoice"></select> s
</wicket:panel>
```

The name of the generic web component, which is used to navigate through slides by the first, next, prior and last links, is AjaxFallbackEntityNavigatePanel. It is an abstract class that requires the implementation of the same two methods as in the AjaxEntitySlideshowPanel component: getNavigatedPanelId and getNavigatedPanel. Note that both components have the same view model in order to share the same current entity. For example, during a slideshow, a user may click on the previous link and the presentation will continue from that slide on.

```
public abstract class AjaxFallbackEntityNavigatePanel extends DmPanel {

	private ViewModel viewModel;

	private View view;

	public AjaxFallbackEntityNavigatePanel(final ViewModel viewModel,
			final View view) {
		super(view.getWicketId());
		this.viewModel = viewModel;
		this.view = view;

		// Link to the first slide.
		AjaxFallbackLink firstLink = new AjaxFallbackLink("firstLink") {
			@Override
			public void onClick(final AjaxRequestTarget target) {
				setCurrentEntity(getFirst());
				replacePanel(target);
			}
		};
		add(firstLink);

		// Link to the prior slide.
		AjaxFallbackLink priorLink = new AjaxFallbackLink("priorLink") {
			@Override
			public void onClick(final AjaxRequestTarget target) {
				setCurrentEntity(getPrior());
				replacePanel(target);
			}
		};
		add(priorLink);
```

```
        // Link to the next slide.
        AjaxFallbackLink nextLink = new AjaxFallbackLink("nextLink") {
            @Override
            public void onClick(final AjaxRequestTarget target) {
                setCurrentEntity(getNext());
                replacePanel(target);
            }
        };
        add(nextLink);

        // Link to the last slide.
        AjaxFallbackLink lastLink = new AjaxFallbackLink("lastLink") {
            @Override
            public void onClick(final AjaxRequestTarget target) {
                setCurrentEntity(getLast());
                replacePanel(target);
            }
        };
        add(lastLink);
    }

    private void setCurrentEntity(IEntity<?> currentEntity) {
        viewModel.setEntity(currentEntity);
    }

    private IEntity<?> getFirst() {
        IEntities<?> entities = viewModel.getEntities();
        return entities.first();
    }

    private IEntity<?> getNext() {
        IEntities entities = viewModel.getEntities();
        IEntity<?> currentEntity = viewModel.getEntity();
        IEntity<?> nextEntity = entities.next(currentEntity);
        if (nextEntity == null) {
            nextEntity = entities.first();
        }
        return nextEntity;
    }

    private IEntity<?> getPrior() {
        IEntities entities = viewModel.getEntities();
        IEntity<?> currentEntity = viewModel.getEntity();
        IEntity<?> priorEntity = entities.prior(currentEntity);
        if (priorEntity == null) {
            priorEntity = entities.last();
        }
        return priorEntity;
    }

    private IEntity<?> getLast() {
```

```
            IEntities<?> entities = viewModel.getEntities();
            return entities.last();
      }

      private void replacePanel(final AjaxRequestTarget target) {
            Panel currentNavigatedPanel = (Panel) getParent().get(
                        getNavigatedPanelId());

            view.setWicketId(currentNavigatedPanel.getId());

            Panel newNavigatedPanel = getNavigatedPanel(viewModel, view);
            newNavigatedPanel.setOutputMarkupId(true);

            currentNavigatedPanel.replaceWith(newNavigatedPanel);

            if (target != null) {
                  target.addComponent(newNavigatedPanel);
            }
      }

      protected abstract String getNavigatedPanelId();

      protected abstract Panel getNavigatedPanel(final ViewModel viewModel,
                  final View view);

}
```

The AjaxFallbackEntityNavigatePanel generic component is a panel. Its logic is based on the current entity obtained from the view model argument. In our case, it is a slide entity. It has the first, next, prior and last links. The AjaxFallbackLink class is used to define each link. If JavaScript [JavaScript] is disabled, a link will behave as a regular link displaying the whole page with the corresponding slide. The private replacePanel method gets the current navigated panel based on the panel id. It then prepares the arguments for the getNavigatedPanel method to obtain the new navigated panel in order to replace the current panel with the new one. The new panel is added conditionally to the Ajax target from the corresponding link method.

The four links in HTML are displayed as button lookalikes.

```
<wicket:panel>
      <a class="button" wicket:id="firstLink">&lt;&lt;</a>
      <a class="button" wicket:id="priorLink">&lt;</a>
      <a class="button" wicket:id="nextLink">&gt;</a>
      <a class="button" wicket:id="lastLink">&gt;&gt;</a>
</wicket:panel>
```

Presentation Editor

A presentation may be edited within the context of one web page. The page is a specific class with the generic EntityEditFormPage name that replaces completely the generic web component with the same

name. The specific page is located in the course.wicket.lecture.presentation package. This page can be reached by signing in, using the *Presentation* link, and clicking on the edit image link of the selected presentation.

In order to support the presentation editor, the new specific property is added to the Point class. The property is type and is used to determine of which type a new point should be so that the proper editor could be displayed. The property is not added in the graphical model to show how a specific property may be added easily later. The type property could have been declared as a derived property in the specific XML configuration, but it is simply added to its class without any additional configuration. As a consequence, the property is not known to Modelibra and its generic code. It will be used locally within the presentation editor.

```
package course.lecture.point;

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

import course.lecture.slide.Slide;

public class Point extends GenPoint {

	...

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

	private String type;

	public void setType(String type) {
		this.type = type;
	}

	public String getType() {
		if (type == null) {
			if (getExplanation() != null) {
				type = "explanation";
			} else if (getUrl() != null) {
				type = "link";
			} else if (getImage() != null) {
				type = "image";
			} else if (getCode() != null) {
				type = "code";
			} else {
				type = "text";
			}
		}
		return type;
	}
```

}

The presentation editor is shown in Figure 13.3.

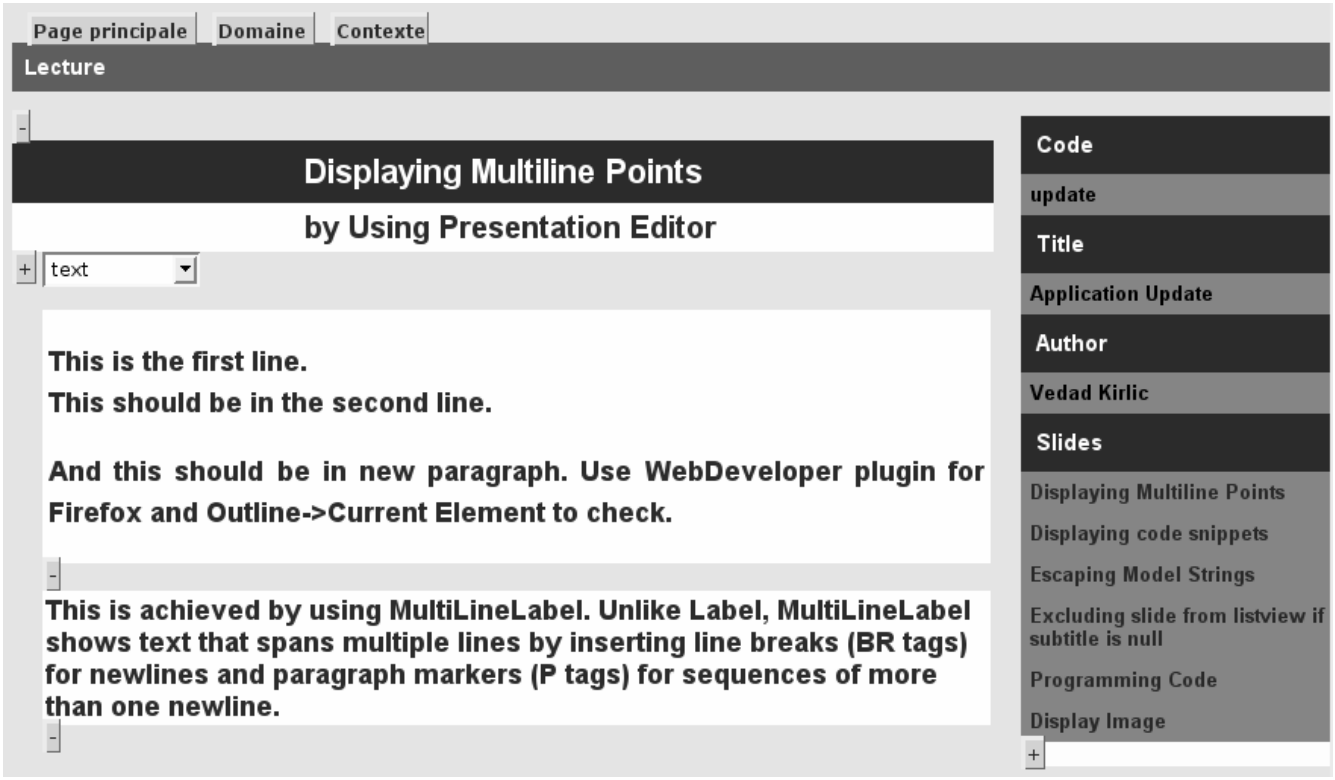

Figure 13.3. Presentation Editor

The Presentation Editor displays the first slide in the content section of the page. The current slide can be changed by selecting the slide title in the *Slides* section in the sidebar. In front of the Slides section, there is the Presentation slides with its essential properties. A slide has a title, a subtitle and a list of points.

The current slide can be removed by the *remove slide* (-) Ajax link. The remove link is located just before the slide title. The slide title can be updated by the Ajax editable label. The title is displayed as a label, until a user clicks on it and a text field appears in the place of the label to allow the user to edit the title. This is a perfect example of the in place component replacement in Ajax where only this section of the page is updated. Without Ajax, the whole page would have been composed on the server again. The slide subtitle can be updated by the Ajax editable label.

A new point may be added to the slide by the *add point* (+) Ajax link. The add link is located just in front of the drop down choice. For a different point type, the dropped down choice would be used before the point is added. A point may be removed from the slide by the *remove point* (-) Ajax link. A point may be edited by simply clicking on it. The point edit section will appear. A click outside the section will accept the change.

The Presentation Editor displays the presentation section with its slides section in the sidebar section of the page. The presentation's code, title and author may be changed by the corresponding Ajax editable labels. The presentation' slides are listed by their titles. A click on the slide title is a click on the Ajax

link that displays the selected slide. A new slide may be added by the *add slide* (+) Ajax link.

Figure 13.3 hints that within the Presentation Editor page there is the Presentation Editor section (panel in Wicket terms), which is decomposed into the Slide Editor section, the Presentation Properties Editor section, and the Presentation Slides section.

The Presentation Editor section is done by the AjaxPresentationEditorPanel component that is added to the specific EntityEditFormPage page.

```
package course.wicket.lecture.presentation;

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;

import course.wicket.lecture.presentation.editor.AjaxPresentationEditorPanel;

public class EntityEditFormPage extends DmUpdatePage {

	public EntityEditFormPage(ViewModel viewModel, View view) {
		super(viewModel, view);

		view.setWicketId("presentationEditor");
		add(new AjaxPresentationEditorPanel(viewModel, view));
	}

	public static PageLink link(final ViewModel viewModel, final View view) {
		PageLink link = new PageLink(view.getWicketId(), new IPageLink() {
			static final long serialVersionUID = 1L;

			public Page getPage() {
				return new EntityEditFormPage(viewModel, view);
			}

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

}
```

The AjaxPresentationEditorPanel component is specific to this application. It contains two specific subcomponents: AjaxSlideListPanel and AjaxSlideEditorPanel. The setOutputMarkupId method used the true argument to enable the Ajax update of the AjaxPresentationEditorPanel component.

There are three Ajax editable labels for the code, title and author required properties of the Presentation

concept. Other properties could have been added here as well. The AjaxEditableLabel component has a Wicket id and a property model. The property model is based on a copy of the presentation and the corresponding property name. When a user clicks on the label it is transformed into a text field. The field value is copied to the property model when the value is entered or when the focus is moved outside the field by clicking somewhere in its context. The presentation is then updated by its copy. If there is an error, it will be displayed in the feedback panel.

```
package course.wicket.lecture.presentation.editor;

import org.apache.wicket.ajax.AjaxRequestTarget;
import org.apache.wicket.extensions.ajax.markup.html.AjaxEditableLabel;
import org.apache.wicket.markup.html.panel.EmptyPanel;
import org.apache.wicket.markup.html.panel.FeedbackPanel;
import org.apache.wicket.model.PropertyModel;
import org.modelibra.wicket.container.DmPanel;
import org.modelibra.wicket.view.View;
import org.modelibra.wicket.view.ViewModel;

import course.lecture.presentation.Presentation;
import course.lecture.presentation.Presentations;
import course.lecture.slide.Slides;
import course.wicket.lecture.slide.editor.AjaxSlideListPanel;
import course.wicket.lecture.slide.editor.AjaxSlideEditorPanel;

public class AjaxPresentationEditorPanel extends DmPanel {

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

		setOutputMarkupId(true);

		final Presentation presentation = (Presentation) viewModel.getEntity();
		final Presentations presentations = (Presentations) viewModel
				.getEntities();

		final FeedbackPanel feedbackPanel = new FeedbackPanel("feedback");
		feedbackPanel.setOutputMarkupId(true);
		add(feedbackPanel);

		final Presentation presentationCopy = presentation.copy();

		add(new AjaxEditableLabel("code", new PropertyModel(presentationCopy,
				"code")) {
			@Override
			protected void onSubmit(AjaxRequestTarget target) {
				presentations.getErrors().empty();
				if (presentations.update(presentation, presentationCopy)) {
					super.onSubmit(target);
				} else {
					addErrorsByKeys(presentations);
```

```
                }
                target.addComponent(feedbackPanel);
            }
        });

        add(new AjaxEditableLabel("title", new PropertyModel(presentationCopy,
                "title")) {
            @Override
            protected void onSubmit(AjaxRequestTarget target) {
                presentations.getErrors().empty();
                if (presentations.update(presentation, presentationCopy)) {
                    super.onSubmit(target);
                } else {
                    addErrorsByKeys(presentations);
                }
                target.addComponent(feedbackPanel);
            }
        });

        add(new AjaxEditableLabel("author", new PropertyModel(presentationCopy,
                "author")) {
            @Override
            protected void onSubmit(AjaxRequestTarget target) {
                presentations.getErrors().empty();
                if (presentations.update(presentation, presentationCopy)) {
                    super.onSubmit(target);
                } else {
                    addErrorsByKeys(presentations);
                }
                target.addComponent(feedbackPanel);
            }
        });

        View presentationSlidesView = new View();
        presentationSlidesView.copyPropertiesFrom(view);
        presentationSlidesView.setWicketId("presentationSlides");

        ViewModel presentationSlidesViewModel = new ViewModel();
        presentationSlidesViewModel.copyPropertiesFrom(viewModel);

        add(new AjaxSlideListPanel(presentationSlidesViewModel,
                presentationSlidesView));

        Slides presentationSlides = presentation.getSlides();
        if (presentationSlides.isEmpty()) {
            add(new EmptyPanel("slideEditor").setOutputMarkupId(true));
        } else {
            add(new AjaxSlideEditorPanel("slideEditor",
                presentationSlides.first(), presentationSlides));
        }
    }
```

```
}
```

The slide editor panel is displayed only if there is at least one slide in the presentation. If there are no slides, the empty panel is shown. The empty panel is enabled for the Ajax update.

The corresponding HTML code for the Presentation Editor is:

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

<html xmlns:wicket>

<wicket:panel>
      <div class="sidebar">
            <div wicket:id="feedback"></div>

            <div class="section-title">
                  <wicket:message key="Presentation.code"/>
            </div>
            <div class="box-subtitle" wicket:id="code"></div>

            <div class="section-title">
                  <wicket:message key="Presentation.title"/>
            </div>
            <div class="box-subtitle" wicket:id="title"></div>

            <div class="section-title">
                  <wicket:message key="Presentation.author"/>
            </div>
            <div class="box-subtitle" wicket:id="author"></div>

            <div wicket:id="presentationSlides"></div>
      </div>
      <div class="content">
            <div wicket:id="slideEditor"></div>
      </div>
</wicket:panel>

</html>
```

The AjaxonSlideListPanel specific component adds an inner class to handle a list of slide titles as Ajax links, and AjaxLink to add a new slide. The onClick method, in the edit slide Ajax link, replaces the current (target) slide editor with the slide editor that corresponds to the list item.

```
package course.wicket.lecture.slide.editor;

import org.apache.wicket.Component;
import org.apache.wicket.ajax.AjaxRequestTarget;
import org.apache.wicket.ajax.markup.html.AjaxLink;
import org.apache.wicket.markup.html.basic.Label;
import org.apache.wicket.markup.html.list.ListItem;
import org.apache.wicket.model.PropertyModel;
```

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

import course.lecture.Lecture;
import course.lecture.presentation.Presentation;
import course.lecture.slide.Slide;
import course.lecture.slide.Slides;

public class AjaxSlideListPanel extends DmPanel {

	public AjaxSlideListPanel(final ViewModel viewModel,
			final View view) {
		super(viewModel, view);
		setOutputMarkupId(true);

		final Presentation presentation = (Presentation) viewModel.getEntity();
		final Lecture lecture = (Lecture) viewModel.getModel();

		final Slides presentationSlides = presentation.getSlides();

		View presentationSlidesView = new View();
		presentationSlidesView.setWicketId("presentationSlideListView");

		ViewModel presentationSlidesViewModel = new ViewModel();
		presentationSlidesViewModel.setEntities(presentationSlides);

		add(new DmListView(presentationSlidesViewModel, presentationSlidesView) {
			@Override
			protected void populateItem(ListItem item) {
				final Slide slide = (Slide) item.getModelObject();

				AjaxLink editSlideLink = new AjaxLink("editSlide") {
					@Override
					public void onClick(AjaxRequestTarget target) {
						replaceSlideEditor(target, slide);
					}
				};
				editSlideLink.add(new Label("slideTitle", new
					PropertyModel(slide, "title")));
				item.add(editSlideLink);

			}

			private void replaceSlideEditor(AjaxRequestTarget target,
					Slide selectedSlide) {
				Component currentSlideEditor = AjaxSlideListPanel.this
						.getParent().get("slideEditor");
				AjaxSlideEditorPanel newSlideEditor = new AjaxSlideEditorPanel(
						"slideEditor", selectedSlide, presentationSlides);
				newSlideEditor.setOutputMarkupId(true);
```

```
                        currentSlideEditor.replaceWith(newSlideEditor);

                        target.addComponent(newSlideEditor);
                  }
            });

            // add slide link
            add(new AjaxLink("addSlide") {
                  @Override
                  public void onClick(AjaxRequestTarget target) {
                        Slide slide = new Slide(lecture);
                        slide.setTitle("Slide " + (presentationSlides.size() + 1));
                        presentationSlides.add(slide);
                        Component currentSlideEditor = AjaxSlideListPanel.this
                                    .getParent().get("slideEditor");
                        AjaxSlideEditorPanel newSlideEditor = new AjaxSlideEditorPanel(
                                    "slideEditor", slide, presentationSlides);
                        currentSlideEditor.replaceWith(newSlideEditor);

                        target.addComponent(AjaxSlideListPanel.this
                                    .getParent());
                  }
            });
      }

}
```

In the onClick event method for the add slide Ajax link, a new slide is constructed and its title is set to the Slide next number. The Slide concept has two required properties, number and title. A value of the number property is auto incremented. The new slide is added and the current slide editor is replaced by the new slide editor. Finally, the presentation editor is updated, by adding to the Ajax target the presentation editor as the parent, to reflect changes both for the list of slides to include the new slide and the slide editor for the new slide.

The corresponding HTML code refers to the list of slide titles as web links and the web link to add a new slide.

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

<html xmlns:wicket>

<wicket:panel>
      <div class="section-title"><wicket:message key="Slides"/></div>

      <div class="box-subtitle" wicket:id="presentationSlideListView">
            <a wicket:id="editSlide">
                  <span wicket:id="slideTitle"></span>
            </a>
      </div>

      <a class="button" wicket:id="addSlide">
```

```
            +
      </a>
</wicket:panel>

</html>
```

The AjaxSlideEditorPanel specific component is provided to remove the current slide, to edit the slide title and subtitle, to select the point type, to add a new point of that type, and to list point editors, one for each point and the point type.

```
package course.wicket.lecture.slide.editor;

import java.util.Arrays;
import java.util.List;

import org.apache.wicket.ajax.AjaxRequestTarget;
import org.apache.wicket.ajax.form.OnChangeAjaxBehavior;
import org.apache.wicket.ajax.markup.html.AjaxLink;
import org.apache.wicket.extensions.ajax.markup.html.AjaxEditableLabel;
import org.apache.wicket.markup.html.form.DropDownChoice;
import org.apache.wicket.markup.html.panel.EmptyPanel;
import org.apache.wicket.markup.html.panel.FeedbackPanel;
import org.apache.wicket.markup.html.panel.Panel;
import org.apache.wicket.model.CompoundPropertyModel;
import org.apache.wicket.model.PropertyModel;
import org.modelibra.wicket.container.DmPanel;

import course.lecture.Lecture;
import course.lecture.point.Point;
import course.lecture.point.Points;
import course.lecture.slide.Slide;
import course.lecture.slide.Slides;
import course.wicket.lecture.point.editor.AjaxPointEditorListPanel;

public class AjaxSlideEditorPanel extends DmPanel {

      private static final List<String> PROPERTY_TYPES = Arrays
                  .asList(new String[] { "text", "code", "explanation", "image",
                              "link" });

      private AjaxPointEditorListPanel slidePointsPanel;

      private String selectedType = "text";

      public AjaxSlideEditorPanel(final String id, final Slide slide,
                  final Slides slides) {
            super(id);
            setOutputMarkupId(true);

            final Slide slideCopy = slide.copy();
            setModel(new CompoundPropertyModel(slideCopy));
```

```
// feedback panel
final FeedbackPanel feedbackPanel = new FeedbackPanel("feedback");
add(feedbackPanel);
feedbackPanel.setOutputMarkupId(true);

add(new AjaxLink("removeSlide") {
      @Override
      public void onClick(AjaxRequestTarget target) {
            Slide priorSlide = slides.prior(slide);

            slides.remove(slide);

            Panel newSlideEditorPanel;
            if (slides.isEmpty()) {
                  newSlideEditorPanel = new EmptyPanel(
                              AjaxSlideEditorPanel.this.getId());
                  newSlideEditorPanel.setOutputMarkupId(true);
            } else {
                  if (priorSlide == null) {
                        priorSlide = slides.first();
                  }
                  newSlideEditorPanel = new AjaxSlideEditorPanel(
                              AjaxSlideEditorPanel.this.getId(), priorSlide,
                              slides);
            }
            AjaxSlideEditorPanel.this.replaceWith(newSlideEditorPanel);
            target.addComponent(newSlideEditorPanel.getParent());
      }
});

AjaxEditableLabel title = new AjaxEditableLabel("title") {
      @Override
      protected void onSubmit(AjaxRequestTarget target) {
            slides.getErrors().empty();
            if (slides.update(slide, slideCopy)) {
                  super.onSubmit(target);
                  target.addComponent(AjaxSlideEditorPanel.this.getParent());
            } else {
                  addErrorsByKeys(slides);
            }
            target.addComponent(feedbackPanel);
      }
};
add(title);

add(new AjaxEditableLabel("subTitle") {
      @Override
      protected void onSubmit(AjaxRequestTarget target) {
            slides.getErrors().empty();
            if (slides.update(slide, slideCopy)) {
                  super.onSubmit(target);
            } else {
```

```
                        addErrorsByKeys(slides);
                    }
                    target.addComponent(feedbackPanel);
                }
            });

            final Points points = slide.getPoints();
            final Lecture lecture = (Lecture) points.getModel();

            DropDownChoice addPointTypeDDC = new DropDownChoice("addPointTypeDDC",
                    new PropertyModel(this, "selectedType"), PROPERTY_TYPES);

            addPointTypeDDC.add(new OnChangeAjaxBehavior() {
                @Override
                protected void onUpdate(AjaxRequestTarget target) {
                    selectedType = (String) getComponent().getModelObject();
                    Point point = new Point(lecture);
                    point.setType(selectedType);
                    point.setText("new " + selectedType + " point");
                    points.add(point);
                    target.addComponent(slidePointsPanel);
                }
            });
            add(addPointTypeDDC.setNullValid(false));

            add(new AjaxLink("addPoint") {
                @Override
                public void onClick(AjaxRequestTarget target) {
                    Point point = new Point(lecture);
                    point.setType(selectedType);
                    point.setText("new " + selectedType + " point");
                    points.add(point);
                    target.addComponent(slidePointsPanel);
                }
            });

            slidePointsPanel = new AjaxPointEditorListPanel("slidePoints", slide
                    .getPoints());
            add(slidePointsPanel);
        }

}
```

The onClick method of the remove slide Ajax link first finds the prior slide, to be able to display it after the current slide is removed. After the slide is removed, if there are no more slides for the presentation, the empty panel is displayed. If there is at least one slide left and if the removed slide was the first slide, the prior slide is set to be the first one. The new slide editor panel is created based on the prior slide. The current slide editor panel is replaced with the new slide editor panel. The parent of the new slide editor is added to the Ajax target to update the parent and in that way all its components. Thus, the slide editor panel is updated as well as the list of presentation slides.

The AjaxEditableLabel widget is used to update both the slide title and subtitle. In the onSubmit method previous slide errors, if any, are emptied. The Wicket model for the slide editor is CompoundPropertyModel, based on a copy of the slides object. Hence, when a slide title is changed, its new value will appear in the copy of the slides object. This copy is used to update the current slide. If the update was successful, the target argument is passed to the inheritance parent of the editable label, and, in the case of the slide title, the parent of the slide editor is added to the target argument to update the list of slide titles. If the update was not successful the errors are added. The feedback panel is added to the target argument to reflect the new status of errors.

A new point may be added by simply selecting the new point type. A point of the same type may also be added by clicking on the add point link.

The HTML code of the slide editor reflects its components.

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

<html xmlns:wicket>

<wicket:panel>
      <div wicket:id="feedback"></div>
      <a class="button" wicket:id="removeSlide">-</a>
      <div class="slide-title" wicket:id="title"></div>
      <div class="slide-subtitle" wicket:id="subTitle"></div>
      <a class="button" wicket:id="addPoint">+</a>
      <select wicket:id="addPointTypeDDC"></select>
      <div wicket:id="slidePoints"></div>
</wicket:panel>

</html>
```

Slide points in the slide editor are handled by the AjaxPointEditorListPanel specific component. This component displays a list of point editors, one for each point. A type of the point editor depends on the point type.

```
package course.wicket.lecture.point.editor;

import org.apache.wicket.ajax.AjaxRequestTarget;
import org.apache.wicket.ajax.markup.html.AjaxLink;
import org.apache.wicket.markup.html.list.ListItem;
import org.apache.wicket.markup.html.panel.FeedbackPanel;
import org.apache.wicket.markup.html.panel.Panel;
import org.modelibra.wicket.container.DmListView;
import org.modelibra.wicket.container.DmPanel;
import org.modelibra.wicket.view.View;
import org.modelibra.wicket.view.ViewModel;
import org.modelibra.wicket.widget.AjaxCodeEditorPanel;
import org.modelibra.wicket.widget.AjaxPropertyEditableMultilineLabel;
import org.modelibra.wicket.widget.AjaxPropertyTextAreaEditableLabel;

import course.lecture.point.Point;
import course.lecture.point.Points;
```

```
public class AjaxPointEditorListPanel extends DmPanel {

	public AjaxPointEditorListPanel(final String id, final Points points) {
		super(id);
		setOutputMarkupId(true);

		View slidePointView = new View();
		slidePointView.setWicketId("slidePointEditorListView");

		ViewModel slidePointViewModel = new ViewModel();
		slidePointViewModel.setEntities(points);

		add(new DmListView(slidePointViewModel, slidePointView) {
			@Override
			protected void populateItem(ListItem item) {
				final Point point = (Point) item.getModelObject();
				String pointType = point.getType();
				final Point pointCopy = point.copy();
				Panel pointEditorPanel;

				final FeedbackPanel feedbackPanel = new FeedbackPanel(
						"feedback");
				feedbackPanel.setOutputMarkupId(true);
				item.add(feedbackPanel);

				View view = new View();
				view.setWicketId("pointEditor");

				ViewModel viewModel = new ViewModel();
				viewModel.setEntity(pointCopy);

				if (pointType.equals("explanation")) {
					viewModel.setPropertyCode("explanation");
					pointEditorPanel = new AjaxPropertyEditableMultilineLabel (
							viewModel, view) {
						@Override
						protected void onSubmit(AjaxRequestTarget target) {
							points.getErrors().empty();
							if (points.update(point, pointCopy)) {
								super.onSubmit(target);
							} else {
								addErrorsByKeys(points);
							}
							target.addComponent(feedbackPanel);
						}
					};
				} else if (pointType.equals("link")) {
					viewModel.setEntity(point);
					viewModel.setEntities(points);
					pointEditorPanel = new AjaxLinkEditorPanel("pointEditor",
							viewModel);
```

```
                        } else if (pointType.equals("image")) {
                              viewModel.setEntity(point);
                              viewModel.setEntities(points);
                              pointEditorPanel = new AjaxImageEditorPanel("pointEditor",
                                          viewModel);
                        } else if (pointType.equals("code")) {
                              viewModel.setPropertyCode("code");
                              pointEditorPanel = new AjaxCodeEditorPanel(viewModel, view)
                              {
                                    @Override
                                    protected void onSubmit(AjaxRequestTarget target) {
                                          points.getErrors().empty();
                                          if (points.update(point, pointCopy)) {
                                                super.onSubmit(target);
                                          } else {
                                                addErrorsByKeys(points);
                                          }
                                          target.addComponent(feedbackPanel);
                                    }
                              };
                        } else {
                              viewModel.setPropertyCode("text");
                              pointEditorPanel = new AjaxPropertyTextAreaEditableLabel (
                                          viewModel, view) {
                                    @Override
                                    protected void onSubmit(AjaxRequestTarget target) {
                                          points.getErrors().empty();
                                          if (points.update(point, pointCopy)) {
                                                super.onSubmit(target);
                                          } else {
                                                addErrorsByKeys(points);
                                          }
                                          target.addComponent(feedbackPanel);
                                    }
                              };
                        }
                        item.add(pointEditorPanel);

                        item.add(new AjaxLink("remove") {
                              @Override
                              public void onClick(AjaxRequestTarget target) {
                                    points.remove(point);
                                    target.addComponent(AjaxPointEditorListPanel.this);
                              }
                        });
                  }
            });
      }

}
```

If the point type is explanation the AjaxPropertyEditableMultilineLabel component is used. If the point

type is link the AjaxLinkEditorPanel component is used. If the point type is image the AjaxImageEditorPanel component is used. If the point type is code the AjaxCodeEditorPanel component is used. Finally, if the point type is text the AjaxPropertyTextAreaEditableLabel component is used. The chosen point may be removed by the remove Ajax link.

The HTML code shows an unordered list, where each list item has the error feedback, the point editor and the remove link.

```
<wicket:panel>
      <ul>
            <li wicket:id="slidePointEditorListView">
                  <div wicket:id="feedback"></div>
                  <div class="slide-point" div wicket:id="pointEditor"></div>
                  <a class="button" wicket:id="remove">-</a>
            </li>
      </ul>
</wicket:panel>
```

The AjaxPropertyEditableMultilineLabel component is used for the explanation point type. It extends the AjaxEditableMultiLineLabel widget from the org.apache.wicket.extensions.ajax.markup.html package. Its sets the new number of columns and rows for the text area. The number of rows depends on the display length of the property in question.

```
package org.modelibra.wicket.widget;

import org.apache.wicket.extensions.ajax.markup.html.AjaxEditableMultiLineLabel;
import org.apache.wicket.model.PropertyModel;
import org.modelibra.wicket.app.DomainApp;
import org.modelibra.wicket.view.View;
import org.modelibra.wicket.view.ViewModel;

public class AjaxPropertyEditableMultilineLabel extends
            AjaxEditableMultiLineLabel {

      public AjaxPropertyEditableMultilineLabel(final ViewModel viewModel,
                  final View view) {
            super(view.getWicketId(), new PropertyModel(viewModel.getEntity(),
                        viewModel.getPropertyCode()));

            int propertySize = viewModel.getPropertyConfig().getDisplayLengthInt();
            int noOfRows = (propertySize / DomainApp.AREA_COLUMN_SIZE) + 1;

            setCols(DomainApp.AREA_COLUMN_SIZE);
            setRows(noOfRows);
      }

}
```

The AjaxPropertyTextAreaEditableLabel component is used for the text point type. It extends the AjaxPropertyEditableMultilineLabel widget. It replaces MultiLineLabel by Label in order to display the text as a label.

```
package org.modelibra.wicket.widget;

import org.apache.wicket.Component;
import org.apache.wicket.MarkupContainer;
import org.apache.wicket.markup.ComponentTag;
import org.apache.wicket.markup.MarkupStream;
import org.apache.wicket.markup.html.basic.Label;
import org.apache.wicket.model.IModel;
import org.modelibra.wicket.view.View;
import org.modelibra.wicket.view.ViewModel;

public class AjaxPropertyTextAreaEditableLabel extends
        AjaxPropertyEditableMultilineLabel {

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

    @Override
    protected Component newLabel(MarkupContainer parent, String componentId,
            IModel model) {
        Label label = new Label(componentId, model) {
            @Override
            protected void onComponentTagBody(MarkupStream markupStream,
                    ComponentTag openTag) {
                if (getModelObject() == null) {
                    replaceComponentTagBody(markupStream, openTag,
                            defaultNullLabel());
                } else {
                    super.onComponentTagBody(markupStream, openTag);
                }
            }
        };
        label.setOutputMarkupId(true);
        label.add(new LabelAjaxBehavior("onclick"));
        return label;
    }

}
```

The AjaxCodeEditorPanel component is used for the code point type. It extends the AjaxPropertyEditableMultilineLabel widget. Its purpose is to provide a different markup tag in the HTML code for the label, so that the programming code is formatted using the pre element.

```
package org.modelibra.wicket.widget;

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

```
public class AjaxCodeEditorPanel extends AjaxPropertyEditableMultilineLabel {

	private static final long serialVersionUID = 1L;

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

}
```

```
<wicket:panel>
	<!-- displayed when component is in display mode -->
	<pre wicket:id="label">[label]</pre>
	<!-- displayed when component is in edit mode -->
	<textarea wicket:id="editor" rows="6" cols="20">[input here]</textarea>
</wicket:panel>
```

The AjaxLinkEditorPanel specific component is used for the link point type. It contains an external link and an edit link to open a modal window to edit the text and url properties.

```
package course.wicket.lecture.point.editor;

import org.apache.wicket.ajax.AjaxRequestTarget;
import org.apache.wicket.ajax.markup.html.AjaxLink;
import org.apache.wicket.extensions.ajax.markup.html.modal.ModalWindow;
import org.apache.wicket.markup.html.link.ExternalLink;
import org.apache.wicket.model.PropertyModel;
import org.modelibra.wicket.container.DmPanel;
import org.modelibra.wicket.view.ViewModel;

import course.lecture.point.Point;

public class AjaxLinkEditorPanel extends DmPanel {

	public AjaxLinkEditorPanel(final String id, final ViewModel viewModel) {
		super(id);

		final Point point = (Point) viewModel.getEntity();

		final ExternalLink link = new ExternalLink("link", new PropertyModel(
				point, "url"), new PropertyModel(point, "text"));
		add(link);
		link.setOutputMarkupId(true);

		final ModalWindow modal;
		add(modal = new ModalWindow("modal"));

		ViewModel modalViewModel = new ViewModel();
		modalViewModel.copyPropertiesFrom(viewModel);
		modalViewModel.setEntity(point);

		modal.setContent(new AjaxLinkModalEditorPanel(modal.getContentId(),
```

```
                        modalViewModel, modal));

            modal.setResizable(false);
            modal.setInitialWidth(70);
            modal.setWidthUnit("em");

            modal.setWindowClosedCallback(new ModalWindow.WindowClosedCallback() {
                  @Override
                  public void onClose(AjaxRequestTarget target) {
                        target.addComponent(link);
                  }
            });

            add(new AjaxLink("edit") {
                  @Override
                  public void onClick(AjaxRequestTarget target) {
                        modal.show(target);
                  }
            });
      }

}
```

The HTML code presents the three components: a modal window, an external link and an edit link.

```
<wicket:panel>
      <div wicket:id="modal"></div>
      <a wicket:id="link"></a>
      <a wicket:id="edit">&</a>
</wicket:panel>
```

The AjaxLinkModalEditorPanel specific component is used as context for the modal window in the AjaxLinkEditorPanel component. It contains a form for updating the text and url properties of the point. The form contains two text field panels. It also has an Ajax button. In its onSubmit method, the point entity is updated and the modal window is closed.

```
package course.wicket.lecture.point.editor;

import org.apache.wicket.ajax.AjaxRequestTarget;
import org.apache.wicket.ajax.markup.html.form.AjaxButton;
import org.apache.wicket.extensions.ajax.markup.html.modal.ModalWindow;
import org.apache.wicket.markup.html.form.Form;
import org.apache.wicket.markup.html.panel.FeedbackPanel;
import org.apache.wicket.model.CompoundPropertyModel;
import org.modelibra.wicket.container.DmPanel;
import org.modelibra.wicket.view.View;
import org.modelibra.wicket.view.ViewModel;
import org.modelibra.wicket.widget.TextFieldPanel;

import course.lecture.point.Point;
import course.lecture.point.Points;
```

```
public class AjaxLinkModalEditorPanel extends DmPanel {

	public AjaxLinkModalEditorPanel(final String id, final ViewModel viewModel,
			final ModalWindow modalWindow) {
		super(id);

		final Points points = (Points) viewModel.getEntities();
		final Point point = (Point) viewModel.getEntity();
		final Point pointCopy = point.copy();

		setModel(new CompoundPropertyModel(pointCopy));

		final FeedbackPanel feedback = new FeedbackPanel("feedback");
		feedback.setOutputMarkupId(true);
		add(feedback);

		Form linkForm = new Form("linkForm");

		linkForm.add(new AjaxButton("submit") {
			@Override
			protected void onSubmit(AjaxRequestTarget target, Form form) {
				points.getErrors().empty();
				if (points.update(point, pointCopy)) {
					modalWindow.close(target);
				} else {
					addErrorsByKeys(points);
					target.addComponent(feedback);
				}
			}

			@Override
			protected void onError(AjaxRequestTarget target, Form form) {
				target.addComponent(feedback);
			}
		});

		add(linkForm);

		View urlView = new View();
		urlView.setWicketId("url");

		ViewModel urlViewModel = new ViewModel();
		urlViewModel.setEntity(pointCopy);
		urlViewModel.setPropertyCode("url");

		TextFieldPanel urlField = new TextFieldPanel(urlViewModel, urlView);
		urlField.getTextField().setRequired(true);
		linkForm.add(urlField);

		View textView = new View();
		textView.setWicketId("text");
```

```
            ViewModel textViewModel = new ViewModel();
            textViewModel.setEntity(pointCopy);
            textViewModel.setPropertyCode("text");
            TextFieldPanel text = new TextFieldPanel(textViewModel, textView);
            text.setAttribute("size", "64");

            linkForm.add(text);
      }

}
```

The HTML code shows the use of the form element and, within the form, the table element with three rows. The first row contains the url field. The second row contains the text field. The third row contains the submit button.

```
<wicket:panel>
      <br/>
      <form wicket:id="linkForm">
            <table>
                  <tr>
                        <th>
                              <wicket:message key="url"/>
                        </th>
                        <td  wicket:id="url">
                        </td>
                  </tr>
                  <tr>
                        <th>
                              <wicket:message key="text"/>
                        </th>
                        <td wicket:id="text">
                        </td>
                  </tr>
                  <tr>
                        <td colspan="2">
                              <input wicket:id="submit" type="submit"
                                    wicket:message="value:save"/>
                        </td>
                  </tr>
            </table>
      </form>
      <div wicket:id="feedback"></div>
</wicket:panel>
```

The AjaxImageEditorPanel class from the course.wicket.lecture.point.editor package is similar to the AjaxLinkEditorPanel component.

Summary

The Cours Presentation web application is introduced. With web the application you can create web slide presentations. A slide has points, where each point is a short text, a formatted explanation text, a web link, an image or a programming code. Images may be uploaded to a server to be used in image point definitions. A presentation may be shown in a slide display with the first, next, prior and last navigation links. There is also a continuous sideshow.

The web application has several specific and generic components developed in Ajax with the help of Wicket. Ajax is not a new technology. It is a combination of old technologies used for creating highly interactive and responsive web pages. A small amount of data is exchanged with a server in such a way to update only a page section. JavaScript is the scripting language in which function calls are made. Those calls are done by Wicket within its Ajax components. Since Ajax components from Wicket are used to develop specific and generic web components in ModelibraWicket, JavaScript function calls are hidden from ModelibraWicket developers. Thus, an object oriented approach to developing Ajax web components can be easily practiced.

The next chapter will explain how inheritance can be used in Modelibra for XML configurations and in Wicket for HTML declarations.

Questions

1. What are technologies used in Ajax?

2. What are advantages of using Ajax web components over regular web components?

3. Look at the JavaScript code generated by Wicket in the Presentation Editor and explain the interaction between a web page on a client computer and a server.

Exercises

Exercise 13.1.

Add the Member and Category concepts to the Lecture model to make presentations categorized and owned by members.

Exercise 13.2.

Consult the spiral.txt file in the project and add a feature that is nice to have.

Exercise 13.3.

Compare the Course Presenation web application with one found at [PreZentit]. Add a feature from PreZentit that is not present in the Course Presenation application, but do it in Ajax.

Web Links

[Ajax] Ajax
http://en.wikipedia.org/wiki/Ajax_(programming)
http://ajaxpatterns.org/wiki/index.php?title=Main_Page

[JavaScript] JavaScript
http://www.w3schools.com/js/default.asp

[PreZentit] PreZentit
http://prezentit.com/