Implement MVC in custom SWT components

Learn how easily Model-View-Controller integrates with Eclipse-based applications.

Tejas R Parajia

Eclipse SWT (Standard Widget Toolkit) offers an extensive set of APIs to implement your custom-made widgets. In this article, the author briefly outlines the MVC (Model-View-Controller) architecture, explains the current implementation of MVC in the form of structured viewers, and shows an implementation using a custom SWT widget.

What is MVC?

The MVC architecture (or design pattern) is a graphical user interface (GUI) design style that consists of three parts: model, view, and controller. MVC decouples presentation from data as well as presentation from operations on that data.

Implementing the MVC architecture differs from other types of applications. The main differences come from how you place and implement business logic or view rendering logic. Unlike typical Web applications, where a programmer would have to design and implement all the MVC parts, Eclipse provides you with an API that does most of the controlling or rendering for you. Hence, one cannot strictly compare Eclipse's MVC implementation with that of the Web or any other application type.

Eclipse JFace

Eclipse JFace implements the MVC architecture using the content provider and label provider. The JFace API wraps the standard (non-trivial) widget, such as table and tree, and implements the structured content providers and label providers. You can implement different content providers based on the widget type. List-oriented viewers will implement the structured viewer, where the content maps to the widget item in a structured (listed) manner.

The base class is called `viewer`, which is an extension of a structured viewer. Viewers act as widget containers. The content provider fetches the data in a structured manner; similarly, the label provider obtains corresponding labels. The JFace viewer implementation retrieves this data, sets the corresponding association, and updates the user interface (UI) component with the data set. It also selects, filters, and sorts.
How to implement JFace

Eclipse View and Viewer perform most of the JFace control functions. Viewer, or the View part of MVC, also acts as a widget container; this is the presentation component.

Eclipse View instantiates viewers, the content provider, and label provider, which acts as the model, holds value objects, and sets them in the Viewer as inputElement.

To create the View, instantiate a Viewer using the createPartControl() method. Listing 1 instantiates a default tree viewer; you can also customize the tree and instantiate the tree viewer by using a constructor with a tree object as the parameter.

Listing 1. CreatePartControl method of ExampleView

```java
public class ExampleView extends ViewPart
{
    ... public void createPartControl(Composite parent)
    { // define a grid layout
        GridLayout layout = new GridLayout();
        layout.numColumns = 1;
        layout.marginHeight = 0;
        layout.marginWidth = 0; 1
        layout.horizontalSpacing = 0;
        layout.verticalSpacing = 1;
        parent.setLayout(layout);
        // create widgets createActionBar(parent);
        createTree(parent); // add context menu and listeners
        viewer.addDoubleClickListener(this); viewer.addSelectionChangedListener(openAction);
        // register viewer so actions respond to selection getSite().setSelectionProvider(viewer);
        hookContextMenu();
    }
    private void createTree(Composite parent)
    { viewer = new TreeViewer(parent, SWT.SINGLE | SWT.H_SCROLL | SWT.V_SCROLL | SWT.BORDER);
        viewer.setContentProvider(new ExampleViewContentProvider()); viewer.setLabelProvider(new ExampleViewLabelProvider());
        viewer.setSorter(new ViewerSorter());
        viewer.setInput(ModelManager.getExampleModel());
        viewer.getControl().setLayoutData(new GridData(GridData.FILL_BOTH));
    } ... }
```

In a separate class, implement ContentProvider, which is an object that provides data to the view using the appropriate interface for your viewer type. For example, you can implement an IStructuredContentProvider or an ITreeContentProvider viewer.

Associate the ContentProvider with the Viewer by implementing one of the following methods in the ContentProvider code:

- getElements(Object parent)
- getChildren(Object element)

Note: The JFace framework will call these methods.

Listing 2. Create a custom ContentProvider

```java
public class ExampleViewContentprovider implements ITreeContentProvide {
    ... }```
The MVC architecture usually involves multiple views and a single data source. Currently, in the Eclipse platform, you can associate only one view with a single model. But you can also create multiple views that can access the same data using an adapter view. Simply include an `inputChanged()` method in the `ContentProvider` class. Whenever the `Viewer` has a new input set, it uses the `inputChanged()` method to notify the `ContentProvider`. The `inputChanged()` method accepts a `Viewer` as an input argument, so multiple views can use a single `ContentProvider`.

**Listing 3. Use the inputChanged method for different Viewers**

```java
/** * Register content provider with model. */
public void inputChanged(Viewer viewer, Object oldInput, Object newInput)
{
    if (newInput != null) {
        this.viewer = viewer;
        this.model = (ExampleDelegate)newInput; this.model.addModelListener(this);
    }
}
```

Using MVC with Eclipse SWT

In most common GUI applications, you create a layout to display the requested data or complete a form, like a UI, to add or modify the data. The example application in Figure 1 shows how you display the data in the custom-made form in read and editable mode from XML storage. It also explains each component's role with respect to the MVC architecture.

**Figure 1. Example application**

![ExampleViewer](image)

Figure 2 shows the application's class diagram to help you better understand the overall architecture.
Creating a control

The `ExampleView` acts as a container of the entire application. It will initialize the application in the `createPartControl` method.

**Listing 4. CreatePartControl method initializes layouts**

```java
public void createPartControl(Composite parent) {
    ExampleEditLayout _layout = new ExampleEditLayout(parent, SWT.NONE, FieldMode.Read, new ExampleViewContentProvider());
}
```

Creating forms and layouts

The base layout class defines the global methods and the declarations that different form applications use. Some container events, which act as call-back mechanisms, are registered here.

**Listing 5. CreateControl method of layout**

```java
public void createControls(int style) {
    GridData gridData;
    Text textFld, subjectFld;
    Control toLabel, ccLabel, bccLabel;
    Control fromDateTime;
    Control control;
    Button durationText;
    Button submit;

    GridLayout layout = new GridLayout(2, false);
}
```
layout.marginWidth = 0;
layout.marginHeight = 4;
setLayout(layout);

//Label
gridData = new GridData(GridData.HORIZONTAL_ALIGN_FILL
| GridData.VERTICAL_ALIGN_CENTER);
gridData.horizontalIndent = 10;
LabelFactory.create(this,
    Messages.getString("ExampleEditLayout.Title"), gridData); //NON-NLS-1$
gridData = new GridData(GridData.HORIZONTAL_ALIGN_FILL
| GridData.VERTICAL_ALIGN_CENTER);
gridData.horizontalIndent = 40;
LabelFactory.create(this, "", gridData);

//Text
gridData = new GridData(GridData.HORIZONTAL_ALIGN_FILL
| GridData.VERTICAL_ALIGN_CENTER);
gridData.horizontalIndent = 10;
control = LabelFactory.create(this,
    Messages.getString("ExampleEditLayout.Email"), gridData); //NON-NLS-1$
gridData = new GridData(GridData.HORIZONTAL_ALIGN_BEGINNING
| GridData.VERTICAL_ALIGN_CENTER);
gridData.horizontalIndent = 10;
control = TextFactory.create(this,
    SWT.BORDER | SWT.V_SCROLL | SWT.WRAP, gridData, FieldMode.Edit); //NON-NLS-1$
addField(new TextField(control, ExampleViewContentProvider.FIRST_INDEX));

//Combo
gridData = new GridData(GridData.HORIZONTAL_ALIGN_FILL
| GridData.VERTICAL_ALIGN_CENTER);
gridData.horizontalIndent = 10;
LabelFactory.create(this,
    Messages.getString("ExampleEditLayout.Group"), gridData); //NON-NLS-1$
gridData = new GridData(GridData.HORIZONTAL_ALIGN_BEGINNING
| GridData.VERTICAL_ALIGN_CENTER);
gridData.horizontalIndent = 40;
control = ComboFactory.create(this,
    FieldMode.Edit, false, gridData); //NON-NLS-1$
addField(new ComboField(control,
    ExampleViewContentProvider.SECOND_INDEX));
    ...

Creating a Field (View)

Field is an abstract class that defines methods to contain various UI controls, along with associated IDs to identify those controls globally. Each UI control will subclass Field and give the content provider read and write ability. Listing 6 creates a Field using the Factory pattern in the Layout class.

Listing 6. Create text objects using the Field class

```java
public class TextField extends Field {
    /**
     * @param control
     * @param id
     */
    public TextField(Control control, int id) {
        super(control, id);
    }

    /* Based on the ID of the widget, values retrieved from
```
* the content provider are set.
*/

```java
public void readFromContent(IExampleContentProvider content) {
    String newText = (String) content.getElement(getId());
    if (newText != null)
        ((Text) _control).setText(newText);
}
/* Based on the ID of the widget, values retrieved from widget are
* sent back to the content provider.
*/

public void writeToContent(IExampleContentProvider content) {
    String newText = ((Text) _control).getText();
    content.setElement(getId(), newText);
}
```

### Simplifying the content provider (Model)

The `ExampleViewContentProvider` acts as a model listener that extends the `IStructuredContentProvider`. It is a simple implementation of the Eclipse API, providing callback to retrieve data. Each item requesting the data is based on the item's unique ID defined in the layout during view creation.

The method call will return the data associated with each global ID defined. In the content provider shown in Listing 7, you can retrieve data from an XML file or from a database using an adapter.

#### Listing 7. Implement a method in a custom ContentProvider

```java
public Object getElement(int iIndex) {
    switch (iIndex) {
    case FIRST_INDEX: return "developer@ibm.com";
    case SECOND_INDEX : return new Integer(1);
    case FOURTH_INDEX : return new Boolean(true);
    case THIRD_INDEX: return new Boolean(false);
    case FIFTH_INDEX: return new Boolean(false);
    }
    return null;
}
```

Once you create the controls and initialize the layout, the Form will ask the content provider to populate the Form controls with the data using the control IDs.

#### Listing 8. Form that initializes Layout and populates controls

```java
public Form (Composite parent, int style, FieldMode mode, ExampleViewContentProvider content) {
    super(parent, style);
    _content = content;
    _style = style;
    setMode(mode);
    init(style);
}

private void init(int style) {
    createControls(style);
    controlsCreated();
}

protected void controlsCreated() {
    readFromContent();
}
```
In conclusion

Web applications were the early implementers of MVC architecture style. However, with the arrival of a simple and powerful application development platform like Eclipse, programmers can easily develop richer UIs in less time and with minimal complexity.
## Downloadable resources

<table>
<thead>
<tr>
<th>Description</th>
<th>Name</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Example code for this article</td>
<td>wa-eclipsemvc_ExampleViewer.zip</td>
<td>33KB</td>
</tr>
</tbody>
</table>
Related topics

- Download a free trial version of WebSphere Application Server Version 6.0.
- Build your next development project with IBM trial software, available for download directly from developerWorks.
- Visit Eclipse.org for comprehensive information on the program and how to use it.
- Read "Getting started with the Eclipse Platform" for a history and overview of Eclipse, including details on how to install Eclipse and plug-ins (developerWorks, November 2002).
- Get Cogito and the Linux kernel source at The Linux Kernel Archives.