

Advanced Widgets for Eclipse

R. Ian Bull, Casey Best, Margaret-Anne Storey
University of Victoria
PO Box 1700 STN CSC
Victoria, BC, V8W 2Y2
Canada
{irbull,cbest,mstorey}@uvic.ca

Abstract

Information Visualization Toolkits are often in the form of applications or complex frameworks and do not integrate into existing applications very easily. In this paper we introduce three Advanced Widgets for Eclipse (AWE) to help create visualizations of complex data. These widgets are packaged as a set of JFace components. The widgets include a Simple UML Class Diagram widget, a Simple Graph Widget and a Nested Zoomable Graph Widget. Each widget conforms to the JFace standards and therefore can easily be incorporated by Java Developers into Eclipse Plug-ins or stand-alone applications.

1 Introduction

Software Engineering promotes code reuse and many widgets sets have been created to support this ideology. Libraries, such as Swing and SWT, are excellent examples of this. There are several GUI libraries available, however, most do not enter the realm of Information Visualization. Most GUI libraries contain components for Lists, Trees, Tables, Rich Text Fields, Buttons, Sliders, etc, but do not provide components for 2D/3D Graphs or UML Diagrams. If the information is complex and needs novel representations, usually the developer has to implement it themselves using 2D and 3D drawing packages (such as Draw2D[1], Piccolo[3], and OpenGL), borrowing concepts from industrial and academic visualization environments such as Tom Sawyer, Snap Together Visualization[8] and SHriMP[7]. This often produces very good tools for very specific domains such as Software Visualization, or very generic visualization environments which fail to be practical for common adoption.

Eclipse is an Integrated Development Environment built as framework with a large selection of plug-ins. Un-

like other IDEs Eclipse is being used for more than just Software Development. Eclipse is being used as an Integrated Learning Environment[10], a L^AT_EX Editor, and even a Platform for Games [5]. In order to increase the productivity of developers contributing plug-ins to the Eclipse environment, or even writing stand-alone applications using SWT, the Eclipse team has packaged a set of common GUI components into an easy-to-use widget set. This widget set consists of common components such as *tables, trees, and lists*, simplifying UI programming tasks by allowing the developer to focus on their own representation of the data instead of worrying about how the UI widget works. This decoupling of data and implementation is achieved by a common interface and a set of data and display providers.

In this paper we introduce three additional components known as the **Advanced Widgets for Eclipse (AWE)**. These components include:

- The UML Widget
- The Graph Widget and
- The Zoomable Nested Graph Widget

Each of these components are available under the Common Public License and are intended to help Eclipse Plug-in Developers create new and innovative visualizations of their data.

1.1 Overview

This paper begins by introducing the AWE the Advanced Widgets for Eclipse (Section 1) and then outlines JFace (Section 2). Section 3 describes each of the three widgets in detail and describes the basic providers for each one. Section 4 shows how one of these widgets can be used to create a simple Java Hierarchy browser. Finally section 5 wraps up with some future work and concluding remarks.

2 JFace

JFace has been designed to wrap common GUI components to make them easier to use. The Eclipse team describes JFace as “A *UI Toolkit that provides classes for developing UI features that can be tedious to implement. JFace operates above the level of a raw widget system.*” Through the use of providers, JFace allows the components to work in a domain independent manner without the developer needing to know the internals of the component. The provider pattern is easy to use, and simply requires the developer to implement the `Provider` interface and pass the instantiated object to the viewer.

For example, the `IContentProvider` interface defines the method:

- `Object[] getElements(Object element)`

The `getElements` method returns a list of objects related to the object `element`.

Providers are created to match the intent of the widget. For example, the *label provider* and the *view sorter*, are used to get display information for an element. The label provider has two basic methods:

- `String getText(Object o)`
- `Image getImage(Object o)`

These methods take the original object as a parameter and return the text and icon for this entity. The following example shows how to implement the `getIcon` method for an application using the object `Gate` to represent an `AndGate`, `OrGate`, `XOrGate`.

```
public Image getIcon( Object o ) {
    Gate g = ( Gate ) o;
    int gateType = g.getGateType();
    switch ( gateType ) {
        case g.ANDGATE:
            return new AndGateIcon();
        case g.ORGATE:
            return new OrGateIcon();
        case g.XORGATE:
            return new XOrGateIcon();
    }
}
```

Listing 1: `getIcon` Implementation

For each of the three widgets described in this paper, a set of providers have been created. These providers have been created in the same spirit as the other JFace providers and match the intent of the widgets. Display providers, such as a `Figure` providers and decorator providers, have also been designed.

3 The Widgets

The Advanced Widgets for Eclipse (AWE) currently consists of three GUI components and a `Figure` project. The components have all been implemented in `Draw2D`[1], a `Drawing` package for SWT. An additional project for graph layout algorithms which work with these widgets is also available from our download site. The `Figure` project is used to give developers a set of `IFigures`, that is `Figures` which implement `Draw2D`'s `IFigure` interface from which to start from. These included `Rectangle` and `Ellipses` with drop shadows and UML Class and Interface `Figures`. The layout algorithm project is available so developers can apply common layout algorithms to their graph structures. Like the rest of the AWE these are data independent. The three GUI components currently available are the UML Widget, the Simple Graph Widget and the Zoomable Widget for Nested Graphs.

3.1 Simple UML Widget

The Simple UML Widget currently only supports UML Class Diagrams. The Content Provider (`UMLContentProvider`) defines the following methods:

- `getTypes(Object type)`
- `getExtends(Object type)`
- `getComposition(Object type)`
- `getAggregation(Object type)`
- `getDependency(Object type)`
- `getAttributes(Object type)`
- `getMethods(Object type)`

Decorators are provided to each `Object` through the `IDecoratorProvider`. This gives the developer the option to set attributes such as *private*, *protected*, *public*, *abstract*, *etc...* on each element. This widget is currently being used in conjunction with the Eclipse's Java Development Tools (JDT) to implement a UML Tool for first year programmers using the Eclipse variant `Gild`[10]. This tool allows students to view the class structure for their assignments and check it against the assignment guidelines.

3.2 Simple Graph Widget

The Simple Graph Widget is used to render two-dimensional graphs. Graphs are considered a set of vertices with a set of edges between pairs of vertices. The developer can specify the `Figure` for each vertex and the `Connector` for each edge. Through the use of

the layout package, different layout algorithms can be applied to the graph. The figures can be any Object which implements Draw2D's IFigure Interface and the Connectors are Objects which implement Draw2D's Connection Interface. The data provider for this widget is the GraphContentProvider which defines the following methods:

- getEntities()
- getOutgoingRelationships(Object vertex)
- getIncomingRelationships(Object vertex)
- getSource(Object edge)
- getDestination(Object edge)

Since a UML Class Diagram is just a graph with a set of pre-defined figures for vertices and edges, we plan to re-implement the UML tool using this canvas. We will keep the same content providers, using an adapter to link the UML content provider to the Simple Graph content provider.

3.3 Zoomable Widget for Nested Graphs

The third widget in the set is a zoomable graph widget is called ZEST (Zoomable Eclipse SHriMP Tool) and was designed by the SHriMP developers. The Nested Zoomable Graph Widget and the Simple Graph Widget can both be used to display information as a graph. However, unlike simple graphs, this component allows a vertex to contain a nested graph. Each of these "nested" vertices can in turn contain relationships to other vertices anywhere in the graph. (See Figure 1). Since vertices can be nested arbitrarily, the depth of the graph can get quite deep. Because of this, this widget includes the ability to "zoom-in" and "zoom-out" to and from any vertex in the graph. The zooming can be animated or instantaneous and the zoom can be centered at a vertex or at the current mouse location. New zoom lenses can also be created and dynamically added to the canvas. Like the previous two components, ZEST is also managed programmatically and the NestedGraphContentProvider is very similar to the Simple Graph Content Provider (Section 3.2). The content provider is actually a sub interface of the simple graph content provider and defines two additional methods:

- getChildren(Object vertex)
- getParents(Object vertex)

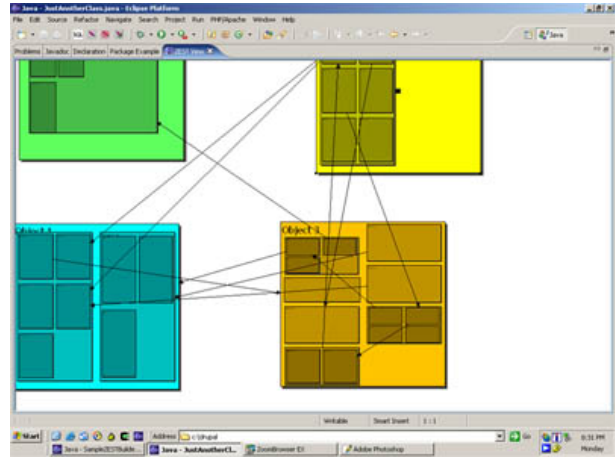


Figure 1: Example of a Nested Graph

4 Sample Application

The AWE has been used to create a plug-in for navigating the type hierarchy of large applications. ZEST is used as the main component, nesting Java files in their package and packages in their parent-packages. Directed relationships are then drawn between subclasses and superclasses such that if file A defines a type which is a subtype of a class in another file (file B) then there is a directed arc from file A to file B. It is presumed that this will help a developer get a sense of how the different packages of a large application are related in regards to their type hierarchy. Since plug-ins like this can be easily created using the AWE, new visualization can be "mocked up" quickly and tested by HCI experts for their strengths and weaknesses.

4.1 Creating the Data Model

The data model for this sample application can be as simple or complex as the developer feels they need. It can be modeled using tools such as EMF[2] or implemented as a single class. The AWE does not presuppose any structure on the data. The only requirement is that the content providers be implemented to bridge the data and the component. In this example we have chosen to model the tool using two classes *Resource* and *Inheritance* and the JDT was used to populate the model.

4.2 Instantiating the Component

The ZEST component can be created in a stand-alone application or as a plug-in for Eclipse. Since we are

```

1 public class PackageView extends ViewPart {
2     private ZestViewer viewer;
3
4     public void createPartControl(Composite parent) {
5         this.viewer = new ZestViewer( parent );
6     } // end of createPartControl
7
8     public void setProject( IProject project ) {
9         PackageWalker packageWalker = new PackageWalker(project);
10        PackageContentProvider pcp = new PackageContentProvider(packageWalker);
11
12        this.viewer.setContentProvider ((NestedGraphContentProvider)pcp);
13        this.viewer.setVertexLabelProvider(new PackageEntityLabelProvider(pcp));
14        this.viewer.setEdgeLabelProvider(new PackageEdgeLabelProvider() );
15        this.viewer.setInput(getViewSite());
16    } // end of setProject
17
18    public void setFocus() {
19        if ( this.viewer != null && this.viewer.getControl() != null )
20            this.viewer.getControl().setFocus();
21    } // end of setFocus
22
23 } // end of PackageView

```

Listing 2: Viewpart Implementation

using the JDT we have decided to build the tool as a plug-in. The ZEST Component is instantiated in the `createPartControl` method of our Viewpart as Listings 2 shows on lines 1-6.

4.3 Populating the Display

The display is populated in the `setProject` method by providing the three providers: `NestedGraphContentProvider`, `EntityLabelProvider` and `EdgeLabelProvider` (Lines 12-14). The `setProject` method is called from an `ViewActionDelegate` and the currently selected project is passed in. The `PackageWalker` (Lines 9-10) uses the JDT to walk the `IProject` structure and populate the model. As mentioned earlier, the three providers implement their respected Provider Interfaces and create a bridge between the model the GUI component. Once these are set, `setInput` is called and this forces the display to update itself.

4.4 Using the Tool

Once the application is completed it will appear as a new view within Eclipse. The view is populated by invoking the `ViewActionDelegate` described above. Once the view is populated (See Figure 1) the nodes will be arranged using a `Grid Layout Algorithm` (the default layout algorithm

for the Zest). The nodes can be moved / resized in the normal way through the use of the mouse. The nodes can be magnified by pressing “Z” and “X” and the zoom mode can be toggled using “Q”. Of course all of these key bindings are configurable.

5 Conclusion and Future Work

By creating GUI components rather than tools, the same complex components can be used for many different applications. We have introduced three new widgets to Eclipse: a `UML Widget`, a `Simple Graph Widget` and a `Zoomable Widget` for `Nested Graphs`. Our toolkit is available for download from <http://www.thechiselgroup.org/>. By creating our toolkit using the same patterns described in `JFace` it is our hope that developers will be able to easily create new visual representations of their data. Many of the same data providers already used in Eclipse can be extended to be used with the AWE which will support the design of new visualizations for IDEs and will support the creation of tools for the Software Process. The work on reusable GUI components for Information Visualization is in its infancy. We plan on looking at several new additions to these components in the coming months in-

cluding: *Additional Content Providers, Composite Arcs, and More Examples.*

5.1 Additional Content Providers

Since there are many ways of representing graphs, we plan on releasing content providers which match these different representations. For example, some represent graphs as an adjacency matrix which defines graphs by knowing for each pair of nodes, whether or not there is an arc connecting them. Others represent graphs as an adjacency list which lists all the arcs in the graphs. The entities can then be derived by looking at the range and domain of the set of arcs. Obviously, if a vertex does not appear in a relationship then it will not be listed but this can be addressed by an additional method which returns all the orphan nodes.

5.2 Composite Arcs

Often graphs are represented at a higher level of abstraction. For example, when viewing architectural diagrams for a large application, it does not make sense to depict each method call. Instead, the relationships between methods are “abstracted” to the level or even the subsystem level. When navigating the data, the user may wish to “drill-down” inside a level to see the actual relationships between the methods, and only at this point should the actual relationships be shown. These types of edges are known as “Composite” or “Lifted” edges [4, 9].

We are currently planning on adding support for such edges directly into our Nested Graph Component.

5.3 Examples

We are currently planning on creating a new application for browsing and viewing relationships in source code using our GUI components. The new application will be modeled after our existing Java Visualization application “Creole” [6], but by using the AWE it will look and feel more like an Eclipse plug-in.

Acknowledgements

We would like to thank IBM Centers for Advanced Studies in both Ottawa and Toronto for their continued support on this project. We would also like to thank Del Myers, Rob Lintern and the rest of the Chisel Group for their help developing this toolkit.

Biographies

R. Ian Bull is a PhD student at the University of Victoria studying under the supervision of Dr. M.A. Storey and working with IBM’s Centers for Advanced Studies. Ian has completed a Master’s Degree in Software Architecture at the University of Waterloo and a Bachelor’s Degree in Computer Science and Software Engineering at the University of Waterloo.

Casey Best is the lead research developer on the ZEST project at the University of Victoria. He has a Masters in Computer Science from the University of Victoria. Casey’s research interests involve overcoming usability issues associated with software architectures. From 2000 to 2002 Casey led the re-engineering of the Shrimp tool at the University of Victoria, which led to the integration of Shrimp with Eclipse (IBM), Protege (Stanford University), and other tools such as CVS.

Dr. Margaret-Anne Storey is an associate professor in the Department of Computer Science at the University of Victoria, a fellow of the British Columbia Advanced Systems Institute (ASI), one of the principal investigators for CSER (Centre for Software Engineering Research) and a visiting scientist at the IBM Centre for Advanced Studies in Toronto. Her main research interests involve understanding how people solve complex tasks, and designing technologies to facilitate the navigation and comprehension of large information spaces

References

- [1] Website. <http://www.eclipse.org/gef>.
- [2] Website. <http://www.eclipse.org/emf>.
- [3] B. B. Bederson, J. Grosjean, and J. Meyer. Toolkit Design for Interactive Structured Graphics. 2004.
- [4] Richard C. Holt. Software Architecture Abstraction and Aggregation as Algebraic Manipulations. In *In Proc of Cascon 1999*, 1999.
- [5] Jos M. Ordax. Website. <http://eclipse-games.sourceforge.net/>.
- [6] Rob Lintern, Jeff Michaud, Margaret-Anne Storey, and Xiaomin Wu. Plugging-in visualization: experiences integrating a visualization tool with Eclipse. In *Proceedings of the 2003 ACM symposium on Software visualization*, pages 47–56, 2009.
- [7] Jeff Michaud, Margaret-Anne D. Storey, and Hausi A. Muller. Integrating information sources for visualizing java programs. In *ICSM*, pages 250–, 2001.

- [8] Chris North and Ben Shneiderman. Snap-together visualization: A user interface for coordinating visualizations via relational schemata. In *Advanced Visual Interfaces*, pages 128–135, 2000.
- [9] Doug Schaffer, Zhengping Zuo, Saul Greenberg, Lyn Bartram, John Dill, Shelli Dubs, and Mark Roseman. Navigating hierarchically clustered networks through she ye and full-zoom methods. *ACM Transactions on Computer-Human Interaction (TOCHI)*, 3:162–188, 1996.
- [10] Margaret-Anne Storey, Daniela Damian, Jeff Michaud, Del Myers, Marcellus Mindel, Daniel German, Mary Sanseverino, and Elizabeth Hargreaves. Improving the Usability of Eclipse for Novice Programmers. In *In Proc. of OOPSLA 2003*, 2003.