**RC22932 (W0310-059) October 9, 2003**
**Computer Science**

# IBM Research Report

## Breaking Out of Eclipse: Developing an ActiveX Host for SWT

**Li-Te Cheng**
IBM Research Division
Lotus Development
1 Rogers St.
Cambridge, MA 02142

**Research Division**
**Almaden - Austin - Beijing - Haifa - India - T. J. Watson - Tokyo - Zurich**

# Breaking out of Eclipse: Developing an ActiveX host for SWT

Li-Te Cheng
IBM Research
Cambridge, MA 02142
li-te_cheng@us.ibm.com

## Abstract

SWT enables Eclipse developers to create user interfaces with native OS look-and-feel and performance. Without significant extra effort, however, SWT applications are limited to operate within the confines of Eclipse, or as a standalone desktop application. This paper describes the motivation and initial efforts to embed SWT applications within an ActiveX container, presents a simple API to enable SWT applications for ActiveX containment, and discusses future directions for embedding the Eclipse framework into other external native OS applications.

## 1 Introduction

SWT is a Java library enabling Eclipse developers to create user interfaces with native OS look-and-feel and performance [4]. It is the foundation of Eclipse's own UI [5], and for various other higher-level libraries, such as JFace [5] and GEF [3]. Developers can use the suite of widgets SWT offers or create their own to produce user interface controls that extend the Eclipse environment, or to support standalone Java applications.

There are alternatives to SWT. A developer could use graphical toolkits like Swing and AWT within Eclipse in a limited fashion [13]. A developer could also embed components to drive a different user interface framework, such as a web browser (e.g. Internet Explorer, Mozilla), or an ActiveX control (e.g. Flash [12]).

A notable aspect of these alternate approaches is their varying ability to be embedded within external non-Java native applications. Swing and AWT use the Java Applet mechanism. Web browsers like Internet Explorer and ActiveX controls [11] can be embedded into native applications directly, often with a few lines of HTML or through an "Insert/Object…" menu option. Embedding can be motivated by a need to reuse components to deliver content in different mediums, such as within a web portal, a spreadsheet, or presentation software.

In contrast, SWT does not offer a direct solution for embedding SWT-based controls into non-Eclipse host applications. This paper focuses on how to embed SWT-based controls in Windows applications through ActiveX, and describes how this is implemented in the native and Java levels. A simple API is then presented to allow developers with no knowledge of ActiveX to enable their SWT components in ActiveX. The paper concludes with a discussion on improvements and directions in embedding the Eclipse framework into other external native OS applications.

## 2 SWT outside of Eclipse

An option to the problem of embedding is to favor a completely non-SWT solution, such as the alternatives described in the previous section. And depending on requirements and goals of the final application, this may be adequate. For example, Java already has an Applet framework, making Swing and AWT a clear choice. The ease of creating interfaces in HTML and the ubiquity of the Web suggests a web-browser based UI. The rich palette of effects, a sizeable development community, and a large base of supported clients are points in favor for Flash.

But if significant effort was already invested in creating an SWT-based application, and if there is a desire not to port the user interface to another alterative, then one must consider how to embed SWT outside of Eclipse. Building on SWT also offers the advantages of integrating with a growing library of other SWT components like those already available in the Eclipse platform and using more powerful and high-level libraries like GEF [3]. For Microsoft Windows developers, SWT also provides mechanisms to incorporate ActiveX controls alongside SWT controls [7]. With these motivations in mind, there are a number of options available to embed SWT code into a native non-Java host application.

One choice is to launch the SWT application as a standalone application from the host application. This involves little or no changes to the SWT

application, and requires some means to launch a new Java process from the host application. The main drawback is the SWT user interface is not truly integrated into the host application from a visual standpoint: it appears as a "pop-up" window. Examples of accomplishing this choice include embedding a system call to execute a batch file, and using Java Web Start [6].

Another option is to abstract out the user interface by some neutral API or language that provides an implementation for SWT, and a medium compatible with the host application. While this is a powerful concept, the drawback here is how much functionality can be expressed and if any performance is sacrificed by this intermediate layer versus pure SWT code. Also, the component may already be written for SWT, and now extra effort must be made to port it into this neutral format. Finally, the intermediate layer must provide an SWT implementation and an implementation compatible with the host application. Examples of neutral layers providing SWT implementations include Luxor [10] and Jelly [1].

Another option is to use the experimental SWT class, org.eclipse.swt.internal.awt.win32. SWT_AWT, to embed SWT within an AWT pane which would then be placed inside an Applet [8]. However, security permissions must be set to make this work, this only works in Windows, and not many applications support direct embedding of applets (unless the applet itself was wrapped by an embeddable web browser control).

One can also use the Java Native Interface (JNI). While many applications use JNI to incorporate native code into Java (for instance, SWT does this internally), JNI does offer an API to create a Java process and execute Java code directly from native code [14]. Like the previous solution, using JNI requires a platform-specific implementation. Also, using JNI does not guarantee a solution to truly embedding the SWT application within its host.

# 3 Creating an ActiveX Host

My work focuses on a JNI-based solution using an ActiveX control [11] to host the SWT component. While limited to the Microsoft Windows environment, an ActiveX control can supply a window handle to embed the SWT component into a host application and is supported by many popular Microsoft Windows based applications such as Internet Explorer, Word, etc. In this section, I describe the process of creating the ActiveX control and enabling it to use SWT.

The first step is to create the ActiveX control itself. This can be accomplished quickly by using the MFC ActiveX Wizard in Microsoft Visual Studio. The ActiveX control I used was based on the STATIC control, and used the "Available in Insert Object dialog", "Activates when visible", and "Acts as a simple frame control" options. This generates the necessary C++ code for a simple ActiveX control consisting of a blank rectangular frame.

Next is to determine where the SWT code to use and supporting classpath information is. In my implementation, I load a text file containing this information from a fixed location.

. The next step is to initiate the JVM from within C++. This is done only once when the ActiveX control is first "activated" in the host application. The OS registry is checked for an installed JVM, and if found, its installation directory is obtained. The JVM's DLL is then located and loaded using the "LoadLibrary" Windows API function. A handle to JNI invocation API function, "JNI_CreateJavaVM", is then found from the loaded DLL. JNI_CreateVM is called to start a JVM thread with classpath and library path parameters [14]. Note that I only link to the JNI invocation API at runtime through DLL entry point calls. Thus, I only need to compile my C++ code with the JNI header file, and not link with a specific JVM's library file.

With the JVM ready, the SWT control can be instantiated and interconnected with rendering events fired on the ActiveX control. JNI's reflection API is used to obtain method handles to the SWT control's default constructor and a set of event handlers conforming to a Java interface specifying how to make an SWT control usable within its ActiveX "container." The ActiveX control's "OnWndMsg" method is overloaded to intercept Windows event calls. Events such as WM_PAINT, WM_SIZE, and WM_MOVE are trapped and the Java code corresponding to these events are called via JNI.

# 4 Enabling SWT for ActiveX

This part describes the steps used to enable an SWT component to work within the ActiveX host described in the previous section. The key is to modify SWT to create a Shell that is a child of an external, non-SWT window.

```
public interface IActiveXControl
{
  public void activeXCreate(
      long window_handle,
      long module_instance,
      int x, int y,
      int w, int h);

  public void activeXResize(
      int w, int h );

  public void activeXMove(
      int x, int y );

  public void activeXPaint();

  public void activeXDestroy();
}
```

**Figure 1:** IActiveXControl, a basic interface defining Java methods called by the ActiveX container when various UI events occur.

In examining the SWT source code, I found that one of its fundamental classes, Control, has a private method, "createHandle", for creating the Windows handle for housing an SWT control. The SWT Shell class, which is a child of the Control class, is used as the top level control of an SWT application, and offers the SWT.win32_new() method to create a Shell as child of an external non-SWT window handle. However, the window creation code in createHandle uses default location and size to initialize the Shell's bounds, and the style bits do not force the window to be an embedded child widget.

Thus, I modified createHandle in the Control class to call a protected method, "customCreateHandle" which by default uses the original SWT implementation of createHandle. Then I added a new class in the SWT package, ActiveXHost, which creates an extended version of Shell (ActiveXHost must be defined in the SWT package set, otherwise I could not subclass the Shell class), which overloads createHandle to create an SWT Shell, but with style bits to force the Shell to be an embedded child widget, and with position and size information supplied by the containing ActiveX control.

Then I created an interface class, IActiveXControl (see Figure 1), which defines the methods called by the ActiveX control to handle a rudimentary set of Window UI events (create, paint, move, resize, destroy). I also created an abstract class, AbstractActiveXControl, which provides a refer-

```
public class
  AbstractActiveXControl
  implements IActiveXControl
{
  Display display;
  Shell shell;

  public
    AbstractActiveXControl() {}

  public abstract void
    createControl(Shell shell);

  public void activeXCreate(
    long hwnd, long inst,
    int x, int y, int w, int h){
    ActiveXHost host = new
      ActiveXHost(hwnd,inst,
        x,y,w,h);
    display = host.getDisplay();
    shell = host.createShell();

    createControl(shell);

    while(!shell.isDisposed()) {
 if (!display.readAndDispatch())
      display.sleep();
    }
    shell.dispose();
  }

  public void activeXResize(
      int w, int h) {
     shell.setSize(w,h);
  }

  public void activeXMove(
      int x, int y) {
      shell.setLocation(x,y);
  }

  public void activeXPaint() {
      shell.redraw();
  }

  public void activeXDestroy() {
      shell.close();
  }
}
```

**Figure 2:** AbstractActiveXControl, a reference implementation of IActiveXControl. It uses the modified version of the SWT library, notably the new class, ActiveXHost, to set up a Shell that is embedded in the ActiveX container.

ence implementation using the new ActiveXHost class I added to the SWT package (see Figure 2). A developer only needs to implement one method in AbstractActiveXControl, "createControl" (see Figure 3 for an example), where he or she can set up the entire component for use in the ActiveX container, and ensure the code is built using the customized version of SWT. Figure 4 shows the final results of embedding SWT components within a web page and a native Windows application.

# 6 Discussion

This work assumes Java and SWT are already installed. Also, only SWT components are supported, not the entire Eclipse platform. A significant improvement is to include a distribution, deployment, security, and execution mechanism for the core Eclipse platform. This can be done by incorporating the work underway at the Equinox project [2], which is already investigating these problems. Another point of reference is the source code responsible for launching the Eclipse platform. These improvements can be executed during the JVM start up process. It may also be desirable to make the Eclipse platform an ongoing process which the ActiveX control can then connect to, rather than instantiating a new JVM and new platform from scratch every time the control is instantiated.

Using an ActiveX container for SWT is similar to using an Applet to host SWT via AWT, but the ActiveX container can be embedded directly in other Windows applications. The container can also be extended with a COM interface to provide an API for native applications to interact with the SWT component, perhaps by dynamically generating the COM interface through JNI reflection calls on the SWT component (Jawin can help with this [9]). Also, the ActiveX container is configurable via a text file (to specify where the SWT code is), and the developer only needs to implement one Java method and compile against a modified SWT jar file. Thus, without any knowledge of ActiveX, internal SWT routines, or JNI, a Java developer can make SWT part of native, non-SWT applications. This work only touches the potential of bringing forth applications outside the Eclipse environment.

```
public class HelloWorld extends
AbstractActiveXControl
{
  public void createControl(
       Shell shell) {
     shell.setLayout(
        new FillLayout() );
     (new Label(shell,0)).
         setText("Hello World");
  }
}
```

**Figure 3:** A simple example of enabling SWT controls to be used within an ActiveX container. The createControl() method is used by AbstractActiveXControl to set up the SWT widget, and the Shell parameter looks like a normal SWT shell, but in fact is a proxy to the ActiveX container.

# Acknowledgements

I would like to thank Seymour Kellerman for his ideas and early discussion, Carl Kraenzel for asking to "make it happen", and John Patterson for allowing me to go off on a tangent with this work.

# About the Author

Li-Te Cheng is a researcher from IBM Research in the Collaborative User Experience group at Cambridge, Massachusetts. He is currently working on enabling collaborative capabilities for small teams of developers in the Eclipse IDE, and also has interests in lightweight shared workspaces, mobile computing, and augmented reality. He can be reached by email at li-te_cheng@us.ibm.com.

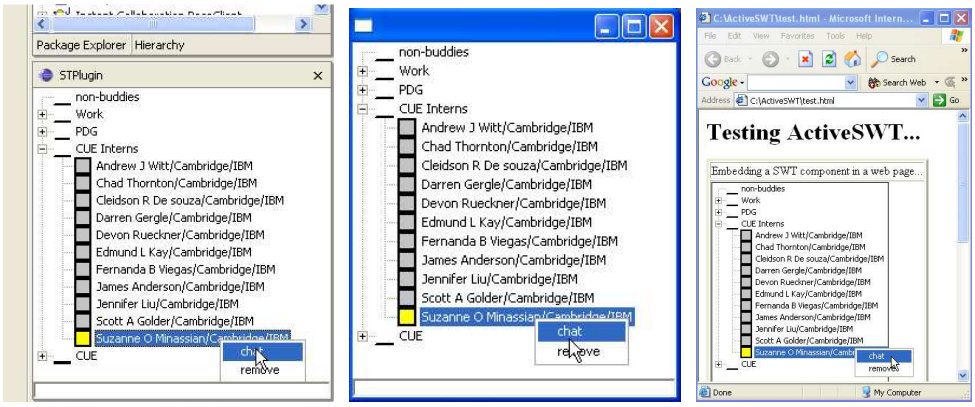
# References

[1] Apache Jakarta Project, Jelly: Executable XML, http://jakarta.apache.org/commons/jelly

[2] Eclipse.org, Equinox Home Page, http://www.eclipse.org/equinox/

[3] Eclipse.org, Graphical Editing Framework, http://www.eclipse.org/gef

[4] Eclipse.org, SWT – Standard Widget Toolkit, http://www.eclipse.org/swt



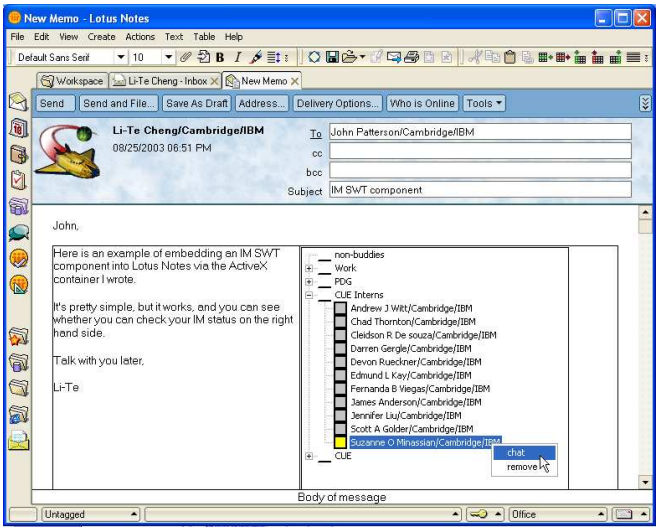




**Figure 4**: Breaking a SWT component out of Eclipse. Top row, left to right: A functional instant messaging buddy list built in SWT using a Tree widget living inside of Eclipse as a ViewPart. Next is the same SWT component, but as a standalone SWT application running inside a pop-up Shell widget. The third screenshot shows the same SWT component embedded in a web page. On the second row, to the left is the same SWT component embedded in a non-Java native application


[5] Eclipse.org, UI – Platform User Interface, http://dev.eclipse.org/viewcvs/index.cgi/%7Echeckout%7E/platform-ui-home/main.html

[6] J. Gunther, Deploy an SWT Application Using Java Web Start, June 19, 2003, http://www-106.ibm.com/developerworks/opensource/library/os-jws/

[7] V. Irvine, ActiveX Support in SWT, March 22, 2001, http://www.eclipse.org/articles/

[8] V. Irvine, Re: How much effort for porting SWT application to applet?, April 4, 2003, http://dev.eclipse.org/newslists/news.eclipse.tools/msg63914.html

[9] Jawin – Interop support for Java, Windows, COM, http://jawinproject.sourceforge.net/

[10] Luxor – XML User Interface Language (XUL) Toolkit, http://luxor-xul.sourceforge.net/

[11] Microsoft, ActiveX Controls, http://www.microsoft.com/com/tech/ActiveX.asp

[12] D. Park, SWT Flash Plugin, http://www.docuverse.com/eclipse/swtflash.jsp

[13] S. Shavor, J. D'Anjou, S. Fairborther, D. Kehn, J. Kellerman, P. McCarthy. Chapter 25: Swing Interoperability. In *The Java Developer's Guide to Eclipse,* pages 545-554, Addison-Wesley , 2003.

[14] Sun, Java Native Interface Specification: The Invocation API, http://java.sun.com/products/jdk/1.2/docs/guide/jni/spec/invocation.doc.html