Vol. 6
Jan '07
www.eclipsemag.net

eclipse MAGAZINE

POWERING THE ECLIPSE ECOSYSTEM

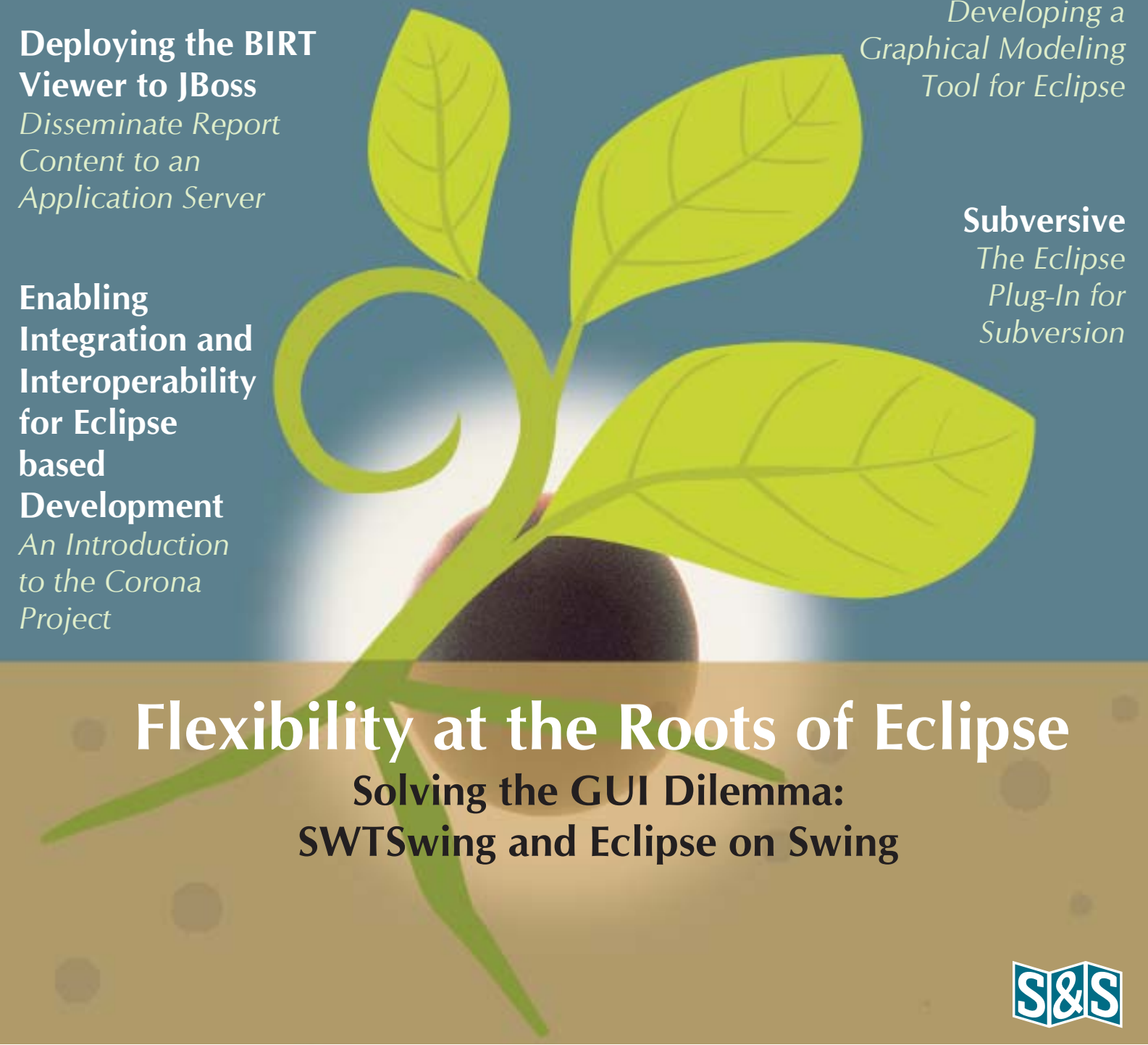
Dynamic Wizard Modeling with GMF
Using GMF to Build a Dynamic Wizard Framework and a Graphical Editor

Introduction to the Generic Eclipse Modeling System
Developing a Graphical Modeling Tool for Eclipse

Deploying the BIRT Viewer to JBoss
Disseminate Report Content to an Application Server

Subversive
The Eclipse Plug-In for Subversion

Enabling Integration and Interoperability for Eclipse based Development
An Introduction to the Corona Project



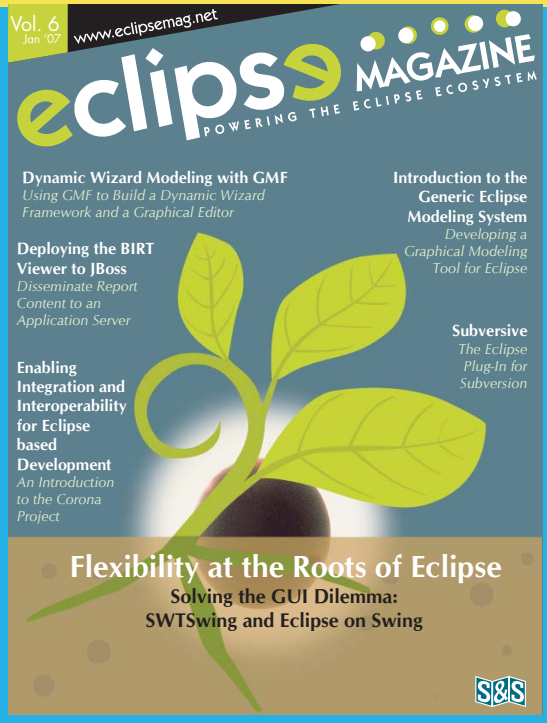
Flexibility at the Roots of Eclipse

Solving the GUI Dilemma: SWTSwing and Eclipse on Swing

S&S

Vol.6
January 2007

Table of Contents



FEATURES

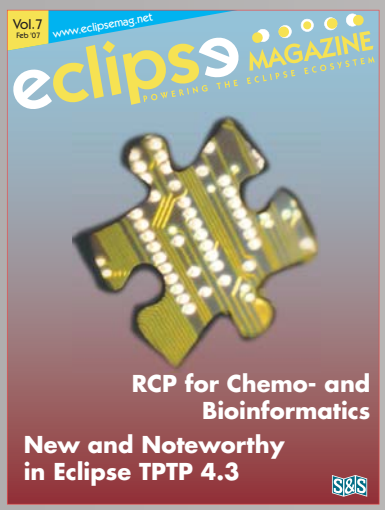
DEPARTMENT

FEATURES

Vol.4
November 2006

Table of Contents

PREVIEW



Flexibility at the Roots of Eclipse

Solving the GUI Dilemma: SWTSwing and Eclipse on Swing

By Dieter Krachtus and Christopher Deckers

No trench in the world of Java is deeper then that between SWT and Swing or Eclipse and Sun. Unity is only found in the knowledge that everybody suffers from this argument. But how to end this almost religious battle over the righteous GUI-toolkit? How to bang their heads together if they only know one point of view—for them or against them! The sister projects SWTSwing and Eclipse on Swing (EOS) achieve this trick. They offer both wranglers a conciliative solution, which combines the best of both worlds and seemingly has only advantages for everybody.

Introduction

The worst choice is to have no choice at all. Therefore, it seems less a curse then a blessing to have this choice with SWT and Swing. However, the lack of time and motivation is often the reason not to master both of the GUI toolkits. Thus, it is not astonishing that the choice of the toolkit isn't focused to the current problem at hand but follows the taste of the developer, which means there was no real choice from the beginning.

Wouldn't it be great, if one had the choice between SWT and Swing not only at the beginning of a project but throughout development? How about using familiar APIs to develop Eclipse-Plug-ins, RCP-applications or a JFace/SWT GUI and still keep the option to switch back and forth between SWT and Swing without changes to your code?

SWTSwing offers this possibility, and **Eclipse on Swing** (EOS) is the proof that it even works for the most complex SWT applications, namely Eclipse itself.

SWTSwing

In August 2005 the SWTSwing project was started but it took over a year until its first official release. The SWT-Snippets [3] are used as developer tests and to document the progress of

This article would be of interest to Technical Managers and Architects, Project Managers and Leads, and Developers. Since this paper is on interoperability, an exposure to multiple technologies (especially SWT, Swing, Eclipse RCP and Plug-ins) is desirable.

AWT, Swing, SWT, JFace, Eclipse RCP

Before taking a closer look at the projects SWTSwing and EOS, it is helpful to remember the capabilities of the solutions used in Java GUI-toolkit development.

- The Abstract Windows Toolkit (AWT) offers the developer only a meagre selection of native widgets (GUI-components like Buttons) to build his GUI. This weakness is due to the lowest-common denominator (LCD) design concept from Sun; that is, only widgets that exist on all Java-platforms were used for the implementation of AWT.
- In contrast, Swing is 100% implemenented in Java and consequently very portable. This advantage, since basically all widgets are emulated, is seen by some as a weakness when it comes to performance. Similarly, the very high flexibility of Swing was achieved at the cost of an often criticised [1] complexity.
- SWT is conceptually similar to AWT, as it uses mainly native widgets. However, emulation is still possible and actually has to be used in order to realize widgets that don't exist on a certain platform. Once exclusively bundled with Eclipse, SWT is now available separately to develop standalone applications. This process has become even more convenient by using the JFace library or the Eclipse RCP.

A more verbose overview about all three UI-toolkits, also touching things like event-handling, is detailed in an IBM developerWorks article, SWT, Swing or AWT: Which is right for you? [2].

the project. These snippets are minimal examples for SWT beginners who want to learn how to build SWT widgets. SWTSwing can execute the majority of these snippets without or only minor bugs by using Swing instead of SWT widgets.

It became obvious that more complex applications and their GUIs were not only more than the sum of their pieces (widgets) but that they also show more bugs than would be expected from their pieces alone when run on Swing.

The sister-project, Eclipse on Swing (EOS) [5], was founded to serve as *Gold Standard* [9] of a complex SWT application for the development of SWTSwing. At the beginning EOS was based on a branched and heavily modified version of the SWTSwing code base, solely to convince Eclipse to start up successfully using Swing.

However, the strategy to use both simple (Snippets) and very complex tests (EOS) was very successful in speeding things up. Error-prone implementations and bugs became visible through the EOS project and led to feedback and changes in SWTSwing.

By now, these improvements of SWTSwing allow the execution of other complex SWT-applications which soon may be used in productive environments. At the moment, a few minor bugs still spoil the overall picture, which however may be already removed when this article is published. To get an impression of the current status of SWTSwing, the most recent screenshots of the popular SWT-applications Azureus [6] and RSSOwl [7] (**Figure 2** and **3**) speak for themselves.

SWT(Swing) Details

Next, let's have a quick look at the general design of SWT in order to understand how SWTSwing is implemented through Swing. SWT offers a public API which has a private interface to the system by using a very thin native layer (see **Figure 3**).

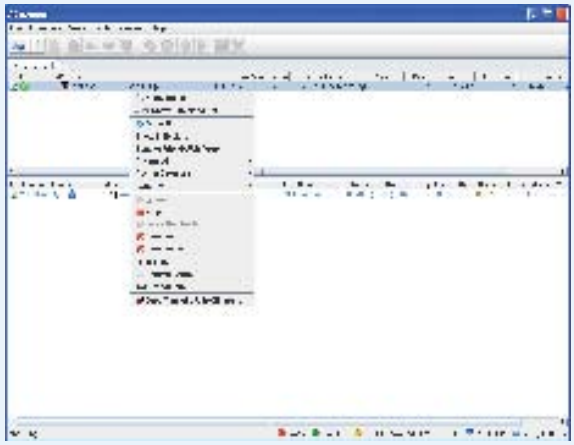


Fig. 1: Azureus

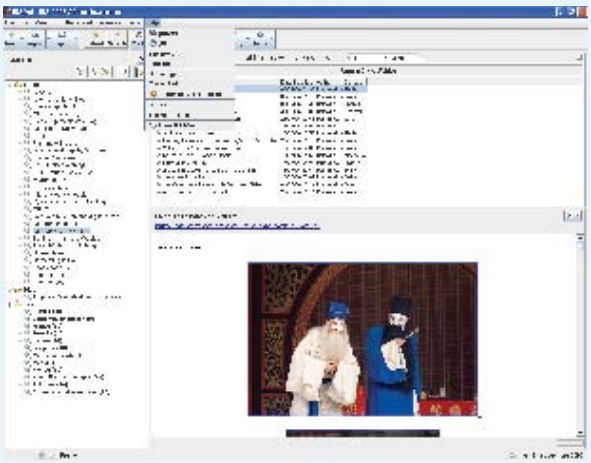


Fig. 2: **RSSOwl**

The actual implementation of SWT follows a general pattern. The basic idea is that all native calls are accessed through a single class called org.eclipse.swt.internal.<given-platform>.OS, which is a one-to-one mapping of the related C-functions. The functions of this very thin layer are directly called by the classes of the SWT public API (see **Listing 1**). Note that AWT also uses a native layer but places some logic inside making it a fatter layer compared to SWT.

In the next step we show how SWTSwing actually creates Swing instead of SWT-widgets. **Listing 1** and **2** show the creation of a Button with SWT and SWTSwing, respectively. The most striking difference for SWTSwing in **Listing 2** is the lack of the class org.eclipse.swt.internal.<given-platform>.OS and thus all native dependencies. Instead the interface org.eclipse.swt.internal.swing.CButton allows creation of instances of the classes CButtonPush, CButtonToggle and CButtonCheck. These classes are specializations of the Swing-classes JButton, JToggleButton and JCheckBox.

For each SWT widget, there exists an interface similar to CButton and classes like CButtonPush, which implements this interface and are derived from Swing widget classes. The core of SWTSwing was technically most difficult to get right, due to a fundamental conceptual difference—**event pumping** is performed explicitly in SWT, whereas Swing manages the event pump internally. To solve that problem, SWTSwing defines two modes of functioning called **best-effort dispatching** and **real dispatching.**

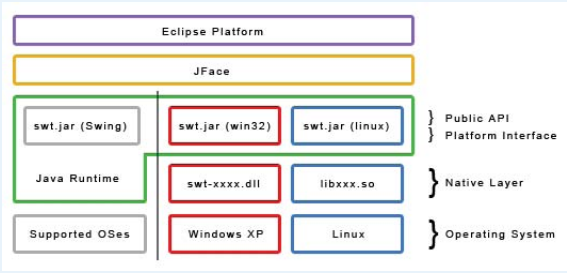


Fig. 3: **How SWTSwing fits**

The real dispatching is the expected SWT behavior, where the SWT UI thread is the operating system UI thread. In case of Swing, that means SWT controls the automatically-managed UI thread. This is possible only if the application or the UI Thread is launched in a certain way, so it is not the default mode, but is recommended for theoretical performance gains and strict compliance with SWT's design.

The best-effort dispatching, which is the default mode, does not control the Swing-UI-thread. It consists of having two UI threads, the SWT and the Swing UI thread, both considered as valid threads for SWT calls. The

Listing 1

```
// Widget-creation in SWT for Windows:

class org.eclipse.swt.widgets.Button extends Control {
  public void setText (String string) {
    // ...
    TCHAR buffer = new TCHAR (getCodePage (), text, true);
    OS.SetWindowText (handle, buffer);
    // ...
  }
}

class org.eclipse.swt.widgets.Control {
  void createHandle () {
    // ...
    // Creation of the handle is done through the super-class
    handle = OS.CreateWindowEx (...);
    // ...
  }
}

class org.eclipse.swt.internal.win32.OS {
  // Native Methods
}
```

Listing 2

```
// Widget-creation in SWTSwing:

class org.eclipse.swt.widgets.Button extends Control {
  Container createHandle () {
    // Creation of the handle takes place within the widget
    return (Container)CButton.Instanciator.
createInstance(this, style);
  }
  public void setText (String string) {
    // ...
    ((CButton)handle).setText(string);
    // ...
  }
}

interface org.eclipse.swt.internal.swing.CButton {
  public static class Instanciator {
    public static CButton createInstance(Button button, int
style) {
      if((style & SWT.PUSH) != 0) {
        return new CButtonPush(button, style);
      }
      if((style & (SWT.CHECK)) != 0) {
        return new CButtonCheck(button, style);
      }
      if((style & (SWT.TOGGLE)) != 0) {
        return new CButtonToggle(button, style);
      }
      // ...
    }
  }
  public void setText(String text);
}

class CButtonPush extends JButton implements CButton
{...}
class CButtonToggle extends JToggleButton implements
CButton {...}
class CButtonCheck extends JCheckBox implements
CButton {...}
```

assumption is that most of the code executed from the SWT thread is some initialization work performed before a Swing window is actually shown and that the rest of the work is performed in response to Swing events. Under that assumption, SWT can act on Swing directly without the risk of an event originating from Swing.

Advantages

One obvious advantage of the Swing implementation of SWT is portability. Separate native libraries are not necessary, which is considered very important by some developers. Moreover, the number of supported platforms increases by those who are exclusively supported by Swing.

The look and feel support also plays a big part, when it comes to 'company branding'. With SWT one is constrained to the native look and feel of the target platform, which in some cases is not what a developer or its company wants.

Another important point for some applications is an easy and flexible deployment. A standalone native SWT application could be deployed with a platform-specific installer, and at the same time without any code changes, as a slender platform-independent application via Webstart.

These are only a few examples where actual problems are solved because you keep the choice of using either SWT or Swing at the same time without any additional work or expense.

Eclipse on Swing

Eclipse on Swing had three motivations for its creation. One role, as mentioned earlier, was to serve as a 'gold standard' for the maturing of SWTSwing. The separation of the standalone code bases of SWTSwing and EOS was reverted recently after a long process of feedback and improvements, whereby now the most up-to-date SWTSwing can be used as a basis for EOS.

The EOS project provides a plug-in that hooks into the Eclipse Preferences and allows a smooth switch between using Swing and SWT. As shown in Figure 4, when using Swing one can naturally select one of the various **Swing Look & Feels** that already come with the plug-in.

It is important to note, that all **Views of EOS** are already usable; only a few minor problems are left that cloud the overall picture during productive work. In Figure 4, Eclipse runs with the Swing Windows Look & Feel, showing hardly any recognizable visual differences to SWT or any other native Windows application.

Another positive feature is the possibility, not only to run Eclipse itself on Swing, but in principle any application based on the Eclipse Rich Client Platform (RCP). Figure 5 shows this capability of the EOS-Plug-in for the JAX 2006 Innovation Award [11] winner Bioclipse [12].

The founders of EOS and SWTSwing currently focus all efforts on improving SWTSwing. The EOS plug-in will be offered in a manually installable form, until a satisfying maturity of SWTSwing and EOS is reached. The plug-in can be downloaded from the EOS project page [5].

With the EOS plug-in the second goal of the EOS project is pushed forward—to provide a non-invasive plug-in to

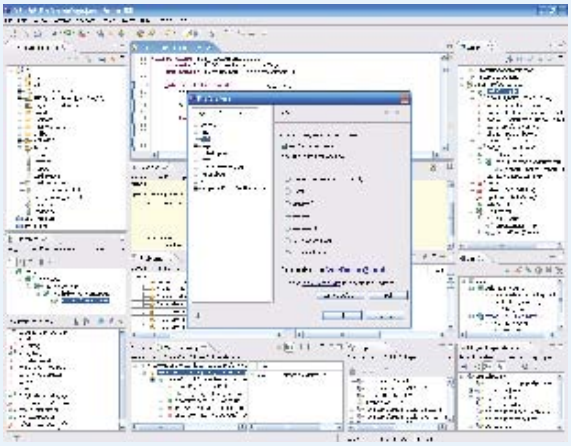


Fig. 4: **Eclipse on Swing**

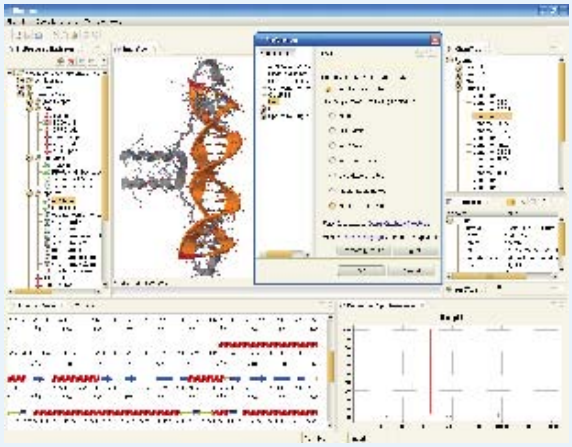


Fig. 5: **Bioclipse**

the Eclipse user, which can be updated from time to time to evaluate and test the progress of SWTSwing and EOS. It could be the key element in the Java community, which mediates between SWT and Swing or Eclipse and Sun, by creating unity and destroying prejudice.

In an interview [8] Mike Milinkovich, the Executive Director of the Eclipse Foundation said: *"One of the nice things about Eclipse is that we are not religious but pragmatic. According to the motto: The people can solve their technical difficulties with this technology. It fits, it functions, it scales. It works like a charm"*.

In the spirit of Eclipse it is now possible to embrace the Swing technology (Figure 4 and 5)

The third, and in a long-term perspective, perhaps the most important motivation for the EOS project is probably not obvious when described with the words—more security and flexibility for the future of Eclipse.

Like a stock market share, the success of Eclipse strongly depends on the fact that many people believe in the future of Eclipse and rely on the promise that Eclipse will still be there in 10 years leading the bleeding edge of technology.

EOS: Eclipse on Spare (Tyre)

Accordingly, it is interesting to examine what are the most fragile parts of Eclipse, in case development has to be carried on by third parties. Firstly, there are many plug-ins for Eclipse with redundancy—meaning that free or commercial plug-ins exist that do more or less the same and which could serve as a replacement. Secondly, Eclipse is almost completely written in Java, well documented, has Unit Tests, and so on. Besides, many developers would be more then qualified to carry on the torch of Eclipse.

The one and only element at the very root of Eclipse, that doesn't fulfill both criteria is **SWT**. Given a worst case scenario there is neither a direct replacement exists for **SWT**, nor is the development trivial due to the native dependencies.

Even in real-case scenarios SWTSwing/EOS could save the day; for example, to port SWT to a new version of an operating system. Swing could deliver a permanent or temporary solution, until the native SWT-implementation has reached a certain stage of maturity.

In the end, it is also less than certain that the decision to go the platform-dependent, native way will be the best solution for Eclipse. At the moment Swing is no better choice for Eclipse than SWT. In ancient times, as an aspirant for the GUI-throne, Swing was once scolded as 'Prince Valium' not without reason. However things turned for the better when it came of age. So who can tell what improvements for Swing a JRE 10.0 has to offer?

Summary

Accepted common sense—namely to choose the technology that solves a problem most elegantly—is usually ignored when it comes to Java GUI development. Rather, once the decision for a GUI toolkit is already cast, arguments and justifications for why this is the most pragmatic decision are found afterwards. The founders of SWTSwing/EOS and

authors of this article stick with Orwell, who reminds us that you usually cannot choose between good and evil but rather the lesser of the two evils.

This means two things. Firstly, there isn't an ultimately best toolkit—SWT and Swing both have their weaknesses. Secondly, from now on, you are in the lucky position to choose between one of these two evils throughout development.

Feedback on this article can be mailed to editors@eclipsemag.net.

Resources & References

[1]
[2] IBM developerWorks: www-128.ibm.com/developerworks/opensource/library/os-swingswt/?ca=dgr-lnxw01WhichGUI
[3] SWT-Snippets: www.eclipse.org/swt/snippets/
[4] SWTSwing: swtswing.sourceforge.net
[5] Eclipse on Swing: eos.sourceforge.net
[6] Azureus: azureus.sourceforge.net
[7] RSSOwl: www.rssowl.org
[8] The Executive Director of the Eclipse Foundation Mike Milinkovich, in Eclipse Magazin Vol. 8
[9] en.wikipedia.org/wiki/Eos
[10] en.wikipedia.org/wiki/Gold_standard_%28test%29
[11] http://jax-award.de/jax_award/index_eng.php
[12] www.bioclipse.net

Dieter Krachtus

is the founder of Eclipse on Swing (EOS) and co-developer of SWTSwing. He works as a developer and consultant with a main focus on Rich Clients on the basis of Swing or the Eclipse RCP. Currently he does a PHD at the Interdisciplinary Center for Scientific Computing (IWR) in Heidelberg.

Christopher Deckers

is the founder of SWTSwing and co-developer of EOS. He lives in France and works as a Senior Java developer. Christopher had always an interest in rich user interfaces, tools and APIs and has his share in many Open source projects.

WHERE DO YOU WANT TO MEET YOUR CUSTOMER?

FOUR SERVICES. ONE SOURCE.

S&S Media

CHAMPIONING CROSS MEDIA COMMUNICATIONS

Software & Support Verlag GmbH
Geleitsstraße 14
60599 Frankfurt am Main, Germany
Phone: +49 (0) 69 63 00 89 0 **Fax:** +49 (0) 69 63 00 89 89
E-mail: sda-asia@software-support.biz
Website:www.software-support.biz

S&S Media
#15/6, I FLOOR, PRIMROSE ROAD,
BANGALORE - 560 025
Off: +91 80 41124392/3 Fax: +91 80 41124391
E-mail: editors@sda-india.com
WebSite:www.softwaresupportmedia.in

S&S MEDIA PTE LTD
133 NEW BRIDGE ROAD #08-10 CHINATOWN POINT
SINGAPORE 059413
TEL: +65 6435 0260 (MAIN LINE) **FAX:** +65 6887 3842
E-MAIL: advertise@sda-asia.com
WebSite:www.softwaresupportmedia.sg

S&S Media
MENARA KADIN INDONESIA,
JAKARTA 12950, INDONESIA
Phone: +62 21 5263083 **Fax:** +62 21 5299 4599
E-mail: contactus@media-ti.co.id
WebSite:www.media-ti.co.id