

Eclipse SWT

or: How I Learned to Stop Worrying
and Love the Bomb

Dr. André Weinand
Eclipse Committer
andre_weinand@ch.ibm.com

Overview

- SWT at a Glance
- the Good, the Bad, ...
- History of SWT
- Design Decisions
- “Philosophies”
- New for 3.0

What is SWT?

- SWT = Standard Widget Toolkit
 - or {Simple, Super, Silly, ...} Widget Toolkit ?

- A portable widget set

- Supported platforms
 - Win32, WinCE
 - Linux/Motif, Solaris/Motif, AIX/Motif, HP-UX/Motif
 - QNX/Photon, Linux/GTK
 - Mac OS X/Carbon

“Hello World” in SWT

```

▪ import org.eclipse.swt.*;
import org.eclipse.swt.widgets.*;

public class HelloWorld {
    public static void main(String [] args) {
        Display display= new Display();
        Shell shell= new Shell(display);
        Label label= new Label(shell, SWT .CENTER);
        label.setText("Hello world");
        label.setBounds(shell.getClientArea());
        shell.open();

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

        display.dispose();
    }
}

```

Why do I like SWT?

- SWT enables applications
 - with native look & feel
 - with excellent UI performance (at least on one platform...)

- Ease of use
 - simple to program
 - small number of abstractions/concepts
 - low mental overhead
 - simple to debug
 - no hidden complexity in natives
 - simple to extend
 - easy access to platform specific features

What I didn't like...

- Windows centric
 - API design
 - Windows first, other platforms later, some very late...
- Strange limitations = “platform realities”
 - non orthogonal
 - limited graphics model
- Makes not real use of OO / Java
 - “style bits”
 - almost no interfaces (exactly one: Listener)
 - no garbage collection for OS resources
 - no add/remove methods

The History ...

- Object Technology International (OTI)
 - virtual machines, class libraries
 - Smalltalk, Java
 - version management systems / IDEs
 - ENVY
 - VisualAge for Java
 - written in IBM/Smalltalk
 - Java runs on Smalltalk VM (!)
 - embedded systems
 - acquired by IBM in 1996

- IBM/Smalltalk
 - CommonWidgets (native, hybrid Smalltalk + OS)
 - Windows, OS/2, Motif, (Mac, OpenLook)

Next Generation: LeapFrog

- Next generation Java IDE (“LeapFrog”)
- Prototype in Java using AWT/Swing
 - appearance and performance problems at the time
 - ➔ felt like a non-native “Java application”
- would AWT/Swing mature fast enough?
- would Java UI development be subject to the same negatives that Smalltalk had faced?

The Dawn of SWT

- SWT was born
 - by porting widgets (CommonWidgets + ExtendedWidgets) from IBM/Smalltalk to Java
- Keep “good” design decisions
 - focus on efficiency, run everywhere ...
- Throw out “bad” design decisions
 - no “X/Motif API”
 - no unnecessary layers ...
- Embrace Java
 - strong typing, visibility ...
 - constructors, overloaded methods ...
 - naming conventions, “good practices” ...

Porting LeapFrog to SWT

- First experiments:
 - LeapFrog debugger was ported to SWT (“Defrogger”)
 - SWT appears as “AWT done right...”
 - but too low level as a replacement for Swing

→ **Splash:** the missing layer

- Application Framework concepts
 - main application windows with menu bar, tool bar, status line
- Actions
 - location-independent user commands
 - contribute action to menu, tool bar, or status line
- Viewers: model aware adapters for SWT widgets
 - trees, tables, lists, styled text, ...

Example: Tree Viewer

- SWT: push model

- ```
Tree tree= new Tree(parent, SWT.H_SCROLL);
TreeItem item= new TreeItem(tree, SWT.NONE);
item.setText("Item 1");
item= new TreeItem(tree, SWT.NONE);
item.setText("Item 2");
```

- Splash: pull model

- ```
Tree tree= new Tree(parent, SWT.H_SCROLL);

TreeViewer tv= new TreeViewer(tree);
tv.setContentProvider(new FileContentProvider());
tv.setLabelProvider(new FileLabelProvider());
tv.setFilter(...);
tv.setSorter(...);
```

Porting LeapFrog to SWT & JFace

- JFace: an IBM Toronto contribution
 - Application framework
 - Dialog, preference, and wizard framework

→ **JFace** = Splash + JFace

- All of LeapFrog was ported over to SWT + JFace
- LeapFrog became **VisualAge Micro Edition (VAME)**

From VisualAge Micro Edition to Eclipse

- VAME didn't replace VisualAge for Java
 - used for embedded system development
 - proprietary code repository (no CVS)
 - missing features, areas for improvement ...
 - no plug-ins
- Improve it, rewrite it, rename it ...
 - another first prototype done in Zurich

→ Eclipse!

- Make it Open Source

What is SWT?

- “The SWT component is designed to provide efficient, portable access to the user-interface facilities of the operating system on which it is implemented.”

from: www.eclipse.org/swt

- The design decisions...
- The philosophies ...

Designed for Maintainability

- API is portable, implementation is not
 - one implementation per platform
 - JARs instead of interfaces
 - most code is platform specific

- Implemented in Java (100% + DLL's)
 - Java used as a system programming language

- Shared Libraries (DLL's) have no interesting code
 - one-to-one mapping of the operating system API
 - leverage operating system programming expertise
 - Java code behaves the same as the equivalent C code

Designed for Efficiency

- Thinnest possible layer over the operating system
 - don't try to “sugar-coat” the operating system
 - minimize the distance to the operating system
 - store as much state as possible in the operating system
 - ➔ no peers
 - ➔ no add/remove methods

Designed for Small Devices

- SWT is small
 - JAR: 800k, DLL: 600k (MacOS X)
 - CLDC (Connected Limited Device Configuration) compliant

- Unused natives can be excluded

- Some packages are optional:
 - `org.eclipse.swt.custom`
 - `org.eclipse.swt.dnd`
 - `org.eclipse.swt.layout`
 - `org.eclipse.swt.accessibility` (stubbed)

SWT Philosophies

- Embrace the OS
 - always use the operating system
 - no free threads
 - no free event loops
 - no free resource management
 - live with the limitations

- Design APIs bottom up
- Use the thinnest possible interface
 - minimize the distance between public API and the operating system calls
- SWT implementation is good example of operating system programming

More SWT Philosophies ...

- Consistent coding style
 - different code takes longer to understand by the team
 - different code is harder to maintain and debug
 - different code gets forgotten, unloved, then hated ...

- Less is more
 - minimize the number of classes, methods, names
 - processor cycles are for the application
 - do you care how SWT is implemented?
 - always take speed/space over some cost to maintainability/flexibility/portability

What's new in SWT 3.0

- Browser Widget
 - Windows: Internet Explorer > 5.0
 - Linux GTK/Motif: Mozilla > 1.4 GTK2
 - MacOS X: Safari/Webkit
 - QNX: Voyager
- Text
 - TextLayout, TextStyle
 - renders complex scripts
 - BIDI
 - multiple fonts
- Tables
 - virtual tables
 - colored cells
- System Tray

More new stuff...

- Non-rectangle windows
- Multiple monitors
- Support for UI testing: generating of low-level events
- 64-Bit Linux GTK port for AMD64
- Accessibility API for custom text widgets
- Support for right-to-left scripts
- System images and cursors

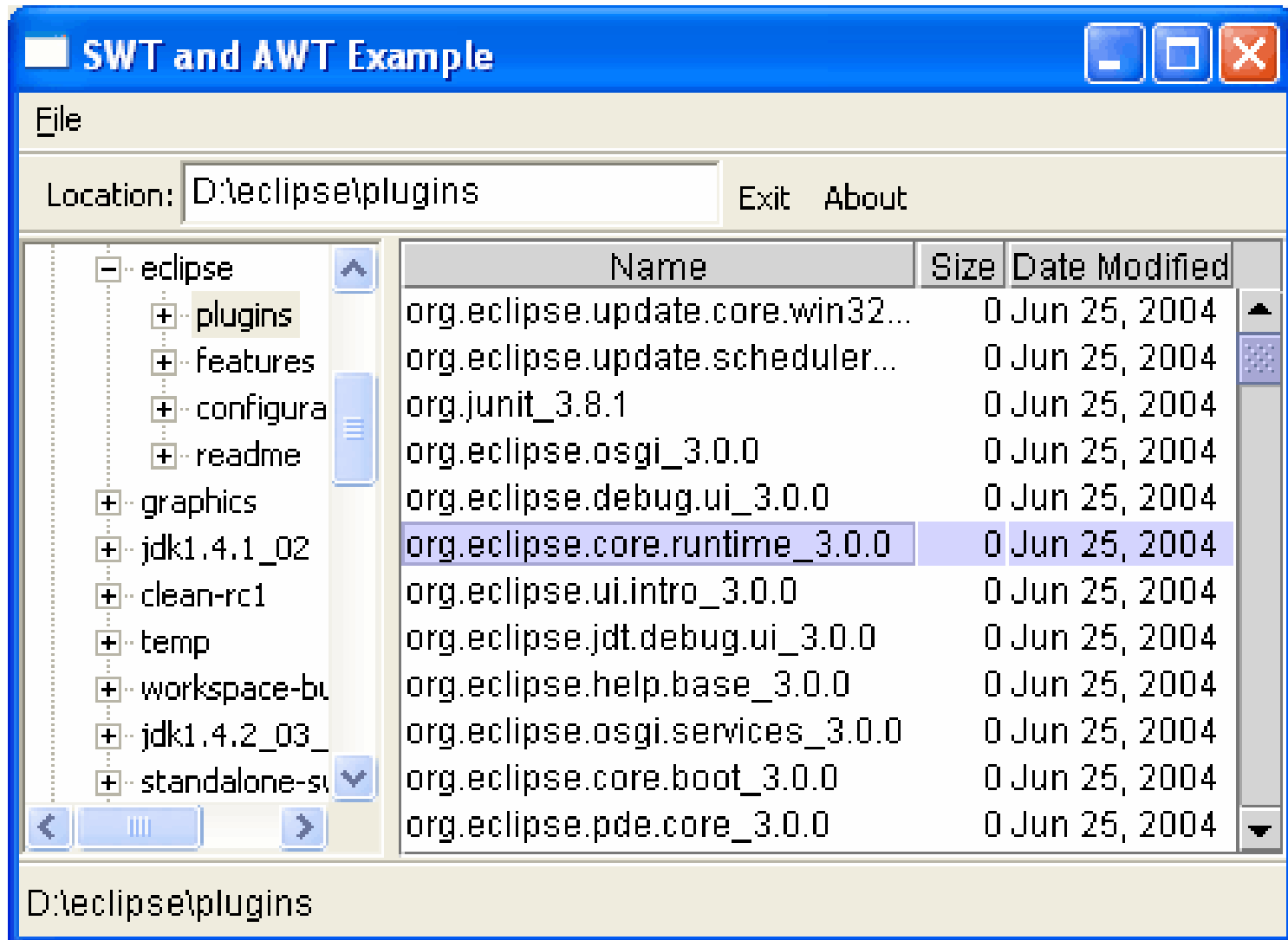
SWT / Swing interoperability

- Embedding AWT/Swing inside SWT supported in
 - Windows with JDK 1.4
 - Linux GTK/Motif with JDK 1.5

- Both directions:
 - Swing inside SWT:
 - `Composite comp= new Composite(shell, SWT.EMBEDDED);`
`java.awt.Frame frame= SWT_AWT.new_Frame(comp);`
`frame.add(new java.awt.TextField());`

 - SWT inside Swing/AWT:
 - `java.awt.Canvas canvas= new java.awt.Canvas();`
`Shell shell= SWT_AWT.new_Shell(display, canvas);`
`Text text= new Text(shell, SWT.NONE);`

Swing inside SWT



Conclusion

- SWT is the state of the art operating system dependent, native widget toolkit

- Use SWT if you care about
 - native look & feel
 - good performance (at least on one platform...)
 - small footprint
 - use of non-portable platform features

- Use Swing if you care about
 - object oriented design and implementation
 - flexibility and configurability
 - OS / platform independence