C++ Gui vs. Java Gui

Daniel Brand, Franz Dietrich

February 9, 2011

Inhaltsverzeichnis

Why talk about GUIs?

- End-user programs often use GUIs
- On low-end machines GUI performance is important
- Is the programming language important?

Backends

- Hardwareacceleration
- Systemcalls

Hardwareacceleration

- 2D
  - copying data (videoram to videoram)
  - draw a solid color
  - draw lines
  - convert mono data to color data
- 3D
  - frontbuffer
  - backbuffer
  - depthbuffer
  - rasterizers
  - texture
- Overlaybuffer
- Hardwaresprites

Systemcalls

- open window
- draw primitives
- draw images
- callbacks and signals

General usage

- request a window
- get a canvas
- insert a layout container
- add widgets (buttons areas etc)
- catch signals

AWT

- threadsafe
- native look and feel
- complex peers (emulate behavior if necessary)
- AWT always should behave the same on every host system

Swing

- extends AWT
- not threadsafe
- lightweight (emulates functions)
- huge code base and hierarchy
- quite flexible to use
- WORE concept (Write once, run everywhere)

Auto dispose

- Swing objects are normal Java objects
- the garbage collector removes them
- you don’t have manage your memory

SWT

- not threadsafe
- heavyweight
- simple peers (just wrappers)
- may behave different depending on host system
- WOTE (write once, test everywhere)

SWT dispose

- you have to dispose every SWT component you create
- if a parent is disposed all children will be disposed as well
- you can optimize your memory usage

What do we expect?

- Swing has to emulate all functions
- SWT uses native functions
- before Java 1.4 Swing was really slow
- Sun worked a lot to minimize this disadvantage
- Swing uses the garbage collector

RAM usage

Open a window with one single button that exits the program.

RAM usage

Win SWT	Win Swing	Linux SWT	Linux Swing
9.2MB	28MB	23.9MB	23.9MB

RAM usage 2

- Swing has a huge code base and a lot to load
- SWT uses the buttons of the host system - Swing has to load its own buttons

Buttons

Add 4000 Buttons to one window and measure the time to resize

Buttons

Win SWT	Win Swing	Linux SWT	Linux Swing
3 sec	< 1 sec	1:04 min	1 sec

Buttons 2

- Swing is quite fast in building its layout
- but Swing can resize the buttons to fit! (1px per button...)
- Swing seems to ignore buttons outside the window
- SWT has more problems to restore its layout

Pixel

Drawing single pixels in different colors

Pixel

Win SWT	Win Swing	Linux SWT	Linux Swing
2 sec	300ms	1307.0 ms	800 ms

Note: Changing the color is a really expensive operation for the hardware

Pixel 2

- Swing uses hardware acceleration on windows
- Changing the color is an expensive operation
- SWT uses GDI+ on Windows, Swing DirectX

Transparent rectangles

Draw 3600 transparent and overlapping rectangles

Transparent rectangles

Win SWT	Win Swing	Linux SWT	Linux Swing
7.4 sec	20ms	562.0ms	15.6 sec

Transparent rectangles 2

- Swing has full hardware acceleration on Windows

Draw text

Draw Text into a drawing area.

Draw text

Win SWT	Win Swing	Linux SWT	Linux Swing
12 sec	250 ms	1507.0 ms	32 sec

Conclusion

- Swing is quite good on Windows
- SWT is not faster than Swing

Sources

- http://www.ibm.com/developerworks/grid/library/os-swingswt/
- http://en.wikipedia.org/
- http://msdn.microsoft.com/en-us/library/ms536334.aspx
- http://www.eclipse.org/swt/

C++ and Gtkmm

What we are going to talk about.

- Introduction to Gtkmm
- architecture
- backends
- performance tests
- comparison

Quick recall

To create a Window in general we have to:

- request a window
- get a canvas
- insert a layout container
- add widgets (buttons areas etc)
- catch signals

Introduction to Gtkmm

OneButton_main.cpp

```
#include "OneButton.h"
#include <gtkmm.h>

int main (int argc, char *argv[])
{
  Gtk::Main kit(argc, argv);

  OneButton win;

  kit.run(win);

  return 0;
}
```

Introduction to Gtkmm

OneButton.h

```
#ifndef GTKMM_EXAMPLE_HELLOWORLD_H
#define GTKMM_EXAMPLE_HELLOWORLD_H

#include <gtkmm/button.h>
#include <gtkmm/window.h>

class OneButton : public Gtk::Window
{

public:
  OneButton();
  virtual ~OneButton();

protected:
  //Signal handlers:
  void on_button_clicked();

  //Member widgets:
  Gtk::Button m_button;
};

#endif // GTKMM_EXAMPLE_HELLOWORLD_H
```

Introduction to Gtkmm

OneButton.cpp

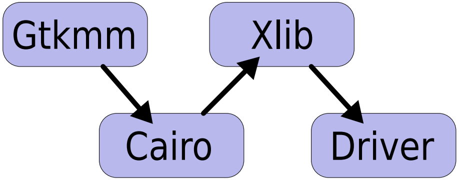
```
#include "OneButton.h"

OneButton::OneButton()
: m_button("Hello␣World")   // creates a new button with label "Hello World".
{
  set_border_width(10);
  m_button.signal_clicked().connect(sigc::mem_fun(*this,
                &OneButton::on_button_clicked));
  // This packs the button into the Window (a container).
  add(m_button);
  m_button.show();
}

OneButton::~OneButton(){}

void OneButton::on_button_clicked()
{
  gtk_main_quit();
}
```

Layers to draw with Gtkmm



Gtkmm

Gtkmm is managing the gui and some other components

- Organize widgets
- Provide standard widgets
- Manage signals
- Provide abstract classes and interfaces to create customized widgets
- Some sort of garbage collection with manage()

Gtkmm

Other uses of this abstractionlayer:

- Provide a rich set of different themes
- Being portable
- Platform independent functions for recently used documents, drag and drop, copy and paste
- Avoid duplication of code

Cairo

Cairo is drawing the gui elements in most cases

- Draw geometric shapes
- Draw lines
- Draw bitmaps
- Draw beziers
- Draw text

XLib

XLib provides an abstraction over the X server protocol

- Manage windows (open, close, move, minimize, maximize, etc.)
- Lowlevel draw abilities (multiple targets)
  - Line
  - Circle
  - Pixel
  - bitmap
- Lowlevel management of capabilities
- Input event generation

Drivers

Drivers provide means to access hardware. They already provide some abstraction but are still very close to the hardware.

- Initialize and setup the hardware
- Drawing in different areas
    - Line
    - Pixel
    - copy memory region

What did we test?

In general we tried to test the different GUI toolkits on different platforms. We had the following testcases.

- RAM usage
- 4000 buttons
- Pixels
- Transparent rectangles
- Drawing Text

RAM usage

Open a window with one single button that exits the program

RAM Usage

Win SWT	Win Swing	Linux SWT	Linux Swing	Linux Gtk
9.2MB	28MB	23.9MB	23.9MB	2.9MB

Buttons

Add 4000 Buttons to one window and measure the time to resize

Buttons

Win SWT	Win Swing	Linux SWT	Linux Swing	Linux Gtk
3 sec	< 1 sec	1:04 min	1 sec	4:07 min

Notes:

- Pool computers did suck a lot less in java SWT (17sec)
    - Could not test gtk on pool computers because of missing libraries
- Windows SWT has a slow reaction to user interaction even when done

Analyse

Analyse the really bad performance of Gtk

- Its not the creation of buttons (which takes 37ms)
- Inserting the Buttons into a scrollable window results in a much faster start and resize
- Drawing seems not to be the problem either (redraw works quite fast)

So my conlusion is that Gtk seems to have a problem with the layouting (when adding a huge list of widgets)

Pixels

Drawing single pixels in different colors

Pixels

Win SWT	Win Swing	Linux SWT	Linux Swing	Linux Gtk
2 sec	300ms	1307.0 ms	800 ms	320ms

Note: Changing the color is a really expensive operation for the hardware

Transparent rectangles

Draw 3600 transparent and overlapping rectangles

Transparent rectangles

Win SWT	Win Swing	Linux SWT	Linux Swing	Linux Gtk
7.4 sec	20ms	562.0ms	15.6 sec	310ms

Notes:

- Result on Linux depends on the driver (Intel appears to have a bug -> demonstration)
- Performance depends on the hardware

Draw Text

Draw Text into a drawing area

Draw Text

Win SWT	Win Swing	Linux SWT	Linux Swing	Linux Gtk
12 sec	250 ms	1507.0 ms	32 sec	2 sec

Conclusion

- Look at the specific needs you have
- There is no good or bad
- with Java Swing it is more easy to write simple GUIs
- Where Ram usage matters take Gtk
- Where maximum independence is needed take Java Swing
- The more performace you want the closer you need to get to the hardware
- Gtk/SWT is more deterministic in its performance

Sources and Links

- http://www.gtk.org/
- http://www.gtkmm.org/en/
- http://cairographics.org/manual/
- Wikipedia
- http://www.x.org/wiki/Development