MOVIAL
IDEAS IN MOTION

Creating a GTK+ based UI's

Markku Ursin / Movial Oy
markku.ursin@movial.fi

San Jose / CELF plenary meeting
Jan 26th, 2005

Agenda

1. About Movial

- Offering:
  - Services – Embedded Linux customer projects; Scratchbox
  - Products – Instant message, presence, and multimedia communication applications
  - Interaction design – Concept design, Usability, User interface design
- Basic facts:
  - Founded in 2001
  - Privately held
  - ~90 employees
  - Based in Helsinki, Finland
- Myself:
  - Employed by Movial since Jan 2003 as a Technical Project manager in the Services unit
  - Before Movial: Speech recognition research

2. GTK+ Technology

1. Introduction
2. History
3. Library structure
4. Theming
5. Pros/Cons

Introduction

“GTK+ is a multi-platform toolkit for creating graphical user interfaces. Offering a complete set of widgets, GTK+ is suitable for projects ranging from small one-off projects to complete application suites.”

- Written in C using an object oriented framework called GObjects
- Used in the GNOME desktop environment
- Language bindings exist for C++, Perl, Python, and others
- License: LGPL
- Features
  - Complete widget (UI component) set
  - Easy to expand by custom widgets
  - Themable
  - Internationalization: support for Unicode and Bi-Di text
  - Input Method API (X11R6 XIM standard)
  - Drag-and-drop
  - Nice GUI builder -- Glade

Introduction...

Options under Linux:

1. Running over X11
2. Running over DirectFB
3. GtkFB

Widget placement

- Widgets are packed into *containers*
  - Containers' contents will be expanded or reduced to fill the container
  - This behavior is controllable
- This makes the UI scalable
- + No need to set fixed pixel values in application code
- - Fulfilling GUI spec pixel values may not be straightforward
  - Take this into consideration when writing the spec

Widgets

- Windows – toplevel and dialog
- Containers – vertical and horizontal boxes, labels
- Buttons, labels, combo box, menus
- Scrollbar, Controlbar
- Animation, Tabs
- Treeview, Listbox
- Etc...

History

- The “GIMP ToolKit”, first versions released in 1997
- The 1.2 version from 2000 is still used in some distributions
- Version 2.0 in 2001
- Current stable version: 2.6.1 (Dec / 2004)

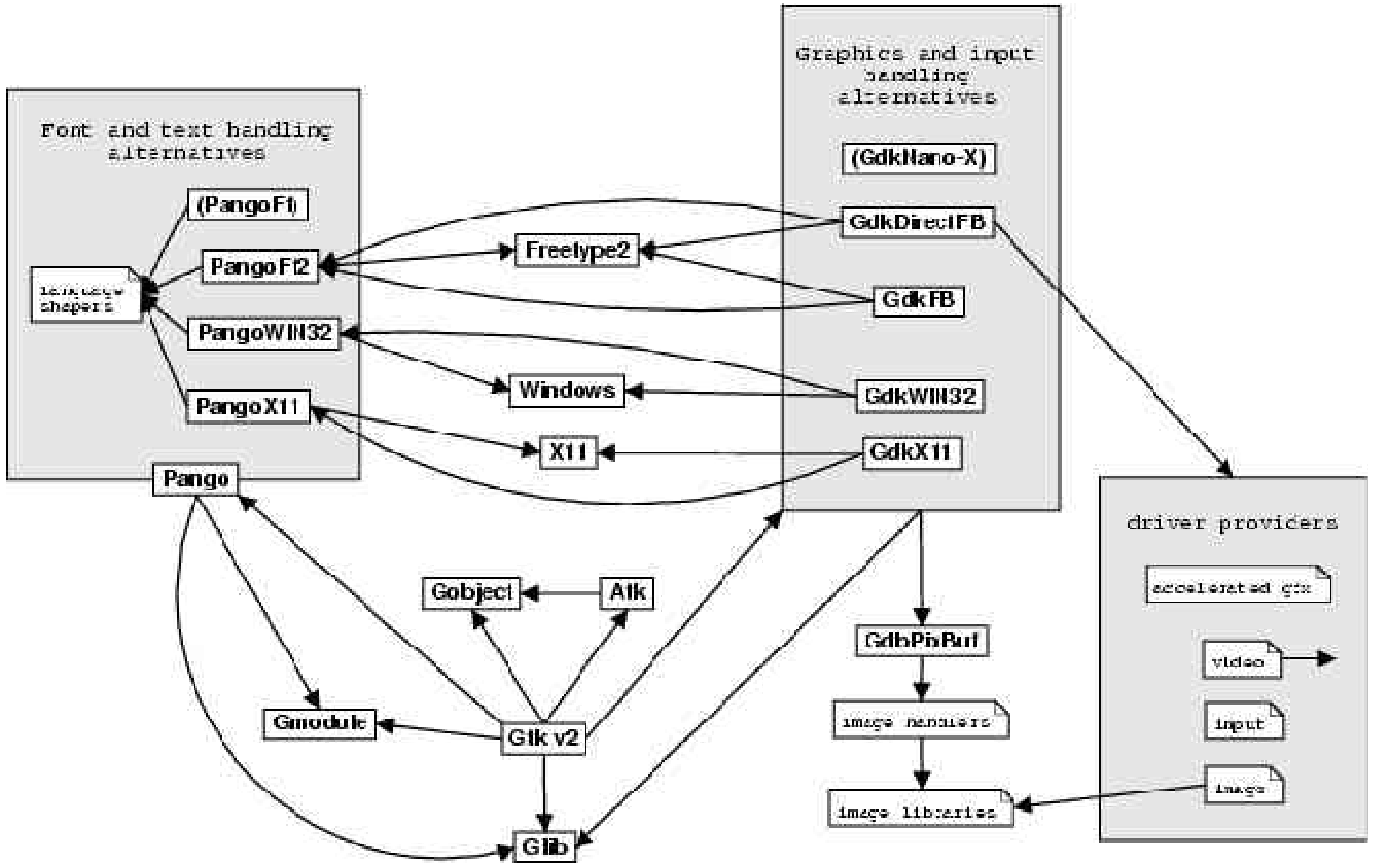
Version differences

- Binary and source compatibility guaranteed between Major versions (e. g. 2.0 and 2.6)
- Major differences between 1.x and 2.0:
  - Better internationalization (Pango)
  - New widgets (TreeView and TextView)
  - API, Graphical, and usability improvements
- New Features in 2.x
  - Fontconfig support – better localization and font matching (2.2)
  - Support for many X extensions (2.2)
  - Widgets: FileSelector, rewritten ComboBox, ToolBar (2.4), IconList (2.6)
  - Unicode 4, Bi-Di improvements (2.4)
  - Icon themes (2.6)
  - Performance improvements

Library Stack

- Libraries are separate projects
- GTK+
  - The UI components
- GDK
  - Thin layer between GTK+ and the windowing system (e. g. X11)
  - Graphics drawing and event handling
- GLib
  - Data structures, Event handling, Utility functions, GObject implementation
- ATK
  - Accessibility features
- Pango
  - Font layout and rendering
- GdkPixbuf
  - Image loading library

MOVIAL


Font and text handling alternatives
(PangoFt)
PangoFt2
language shapers
PangoWIN32
PangoX11
Pango
FreeType2
Windows
X11
Graphics and input handling alternatives
(GdkNano-X)
GdkDirectFB
GdkFB
GdkWIN32
GdkX11
driver providers
accelerated gfx
video
input
image
GdkPixBuf
image handlers
image libraries
Gobject
Atk
Gmodule
Gtk v2
Glib

MOVIAL

Theming

- Using a different *theme*, the looks of an application is radically changed – good for differentiating the product
- Can be changed at runtime
- The theme consists of
  - An RC-file
  - Set of images
  - Theme engine
- The theme controls
  - Colors, icons
  - Fonts
  - Widget specific style properties (border widths, or even behavior)
- Theming should be taken into consideration when implementing own components
  - Implement customizable features as GTK+ style properties

GTK+ Advantages

- Complete widget set
- Scalable UI
- Easily themed
- Easily expanded
- Full internationalization
- Strong OS community
- Stable

GTK+ Issues

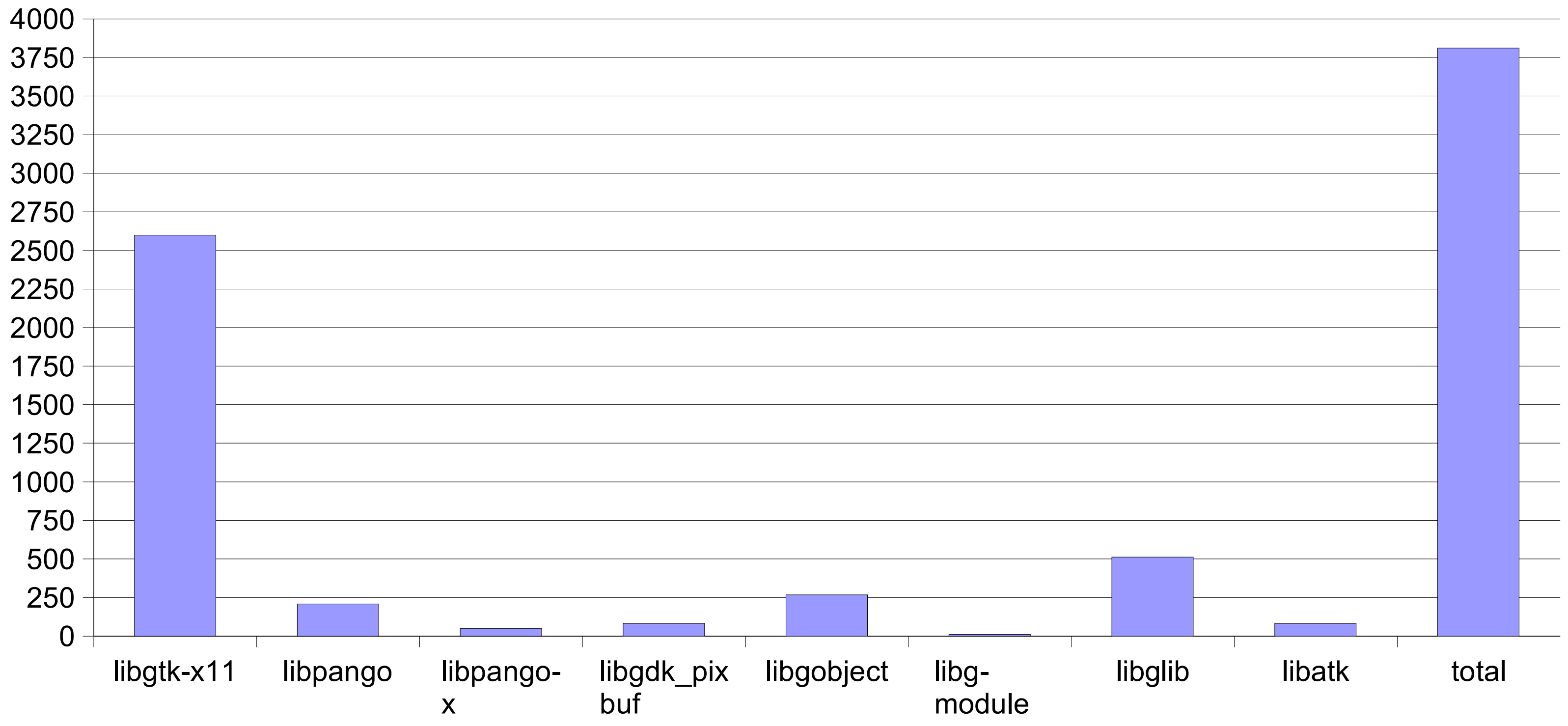
- No ready-made embedded configuration available
- Unfamiliar programming environment
  - GObject framework
- Possibility to get correct-looking results with wrong code
- A lot of even 'stable' open source GTK+ programs spit out a lot of assertions during 'normal' operation
  - One needs to be very careful with type casts etc. as the compiler doesn't check them for you (in C)
- There is a helper application (GOB) for creating GObjects (e. g. widgets), but the licensing approach of that is not clear

3. Embedded Concerns

1. Binary size
2. Memory consumption
3. Performance
4. Development cycle

Binary size

- GTK+/X11 library stack binary sizes:
  - Stripped ARM binaries, version 2.0
- Dependencies (libX11, libm, libc, etc.) form another 2.5 MB



Memory consumption

- Memory consumption of the Glade program on ARM is about 5.5 megs virtual / 3.6 resident
- The Tiny-X server uses 3.9 MB Virtual / 2.7 MB Resident at the same time
- Add another 0.5 MB for the window manager

Performance

- Large widget set, HW requirements not small
- An ARM processor at 200 MHz runs GTK+ neatly (depends on screen size)
- Issues:
  - Application start time
  - Opening new windows
  - Floating point operations
    - *Especially in GdkPixbuf scaling*

Development cycle

- Uses Autotools (autoconf, automake)
- Big library => compiling natively is slow
- Lot's of dependencies => hard to configure for cross compilation
- Compiling natively OK, if you only compile once...
- When modifying the library itself (or developing any application...)

  Scratchbox (http://www.scratchbox.org/) becomes handy

  Shorter development cycle

  Easy to compile add further OS components
- Valgrind is an excellent tool to detect memory leaks and errors
  - only runs on X86

4. GUI Platform Creation Process

1. Requirements specification
2. Technology choice
3. GUI specification
4. Implementation
5. Testing

Naturally, this process is somewhat iterative

Requirements specification

- What kind of device? Set-top-box? Portable?
- What applications are there?
- Which locales need to be supported?
- For whom the device is targeted?
- Use open source applications or develop your own?
- Who gets to install applications?
- Input device
  - Keyboard / Remote
  - Pointer device
  - Touch screen
- Screen size
- HW restrictions

Technology choice

- Which toolkit?
  - GTK
  - Qtopia
- Features
- Licenses
- Present knowledge

If GTK+ is chosen:

- X11 or DirectFB
- Which X server?
- Which window manager
  - Takes care of decoration and windowing policies
- Theme engine

GUI specification

- What functionality is needed by the application(s)
  - Are the native GTK+ widgets enough (they should be!)?
- How should the GUI look like?
  - Make the design so that it is easy o theme
  - Do not hard-code widget sizes etc.
- Create guidelines for application GUI design
- Do the application design according to this guideline
- Check with the community if your needs would fit the plans of the community –> save in maintenance costs

Implementation & Testing

- Get familiar with the toolkit – read the documentation and investigate the source
- Implement custom widgets
- Design with MVC paradigm
- Develop your application
- Test it
  - Graphical testing
    - *Automation with Xnee scripts*
  - Memory testing
    - *Valgrind*
  - Unit testing

Q&A

? => !

Thanks!

For further information, don't hesitate to contact me:

markku.ursin@movial.fi

MOVIAL