GUADEC 2004

GUADEC 2004

Proceedings of the 5th annual GNOME User and Developer European Conference (GUADEC) Kristiasand
28th – 30th June 2004

Kristiansand 2004

Summary
This publication is a collection of papers presented at 5th annual GNOME User and Developer European Conference (GUADEC). GUADEC 2004 brings developers, GNOME Foundation leaders and individual, business and government GNOME and open source software users to Agder University College http://www.hia.no/english/ in Kristiansand, Norway <http://2004.guadec.org/images/norway.png>. The conference is a unique forum for highlighting the capabilities and direction of GNOME, the user environment for desktops, networked servers and portable Internet devices. GUADEC is also featuring meaningful discussions of the future direction of Open Source development.

Publication Series No 107

113 pages
110,00 NOK

ISSN: 1503-5174
ISBN: 82-7117-529-7

Keywords:

- GNOME
- Linux desktop
- Free software
- open source
- usability
- typography
- vector imaging
- user interface development
- rendering
- desktop software
- GTK
- widgets
- accessibility

Table of Contents

BOF: GNOME System Tools
Carlos Garnacho

To input many languages
Tokunaga Hiroyuki

GNOME in Japan
Yukihiro Nakai

GNOME & Usability in Military Systems
Kathy Fernandes

GNOME Roadmap
Dave Camp

GUADEC

5th Annual GNOME Users and Developers European Conference
Kristiansand, Norway
June 28th – 30th 2004

GUADEC is organized by

GNOME Foundation, 8 Cambridge Center,
Cambridge, Massachusetts 02142 U.S.A.
http://foundation.gnome.org
and
Agder University College
http://www.hia.no
(publisher of those proceedings)
in co-operation with Filonova, a company owned
by Agder University College and Adgerforskning.

GNOME Foundation

GUADEC 2004 Sponsors

Cornerstone
Novell

Platinum
HP
Red Hat
Sun

Silver
Bibliofil
Høgskolen I Agder
O' Reilly
Vest-Agder Fylkeskommune
Aust-Agder Fylkeskommune
SAS Braathens

Media Sponsors
Fluendo
Open Surce Development Network (OSDN)
Linux Magazine

gDesklets
an advanced architecture for applets

Martin Grimme <martin@pycage.de>
Christian Meyer <chrisime@gnome.org>

June 11, 2004

1 Introduction

gDesklets[1] provides an advanced architecture for applets and similar things for, but not limited to, the GNOME desktop. This includes desktop applets as well as panel applets or web applets (remote applets). Figure 1 shows a typical gDesklets-enhanced desktop.

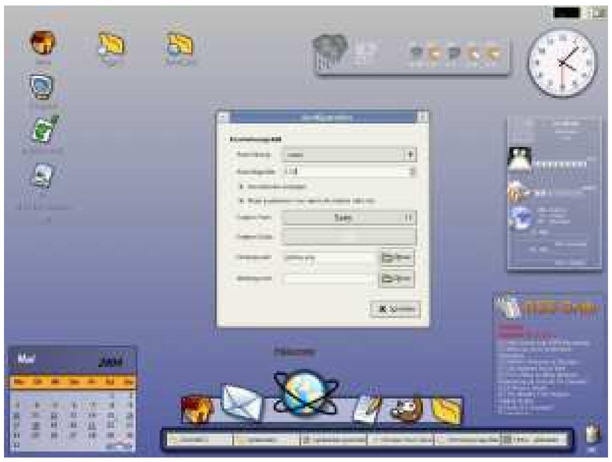


Figure 1: gDesklets on the desktop.

In this talk, we will give you an overview of the framework behind, show you how to use it, and what you can expect for the future.

[1] `http://gdesklets.gnomedesktop.org`

2 Components of the Framework

The gDesklets framework consists of several components, as can be seen in figure 2.

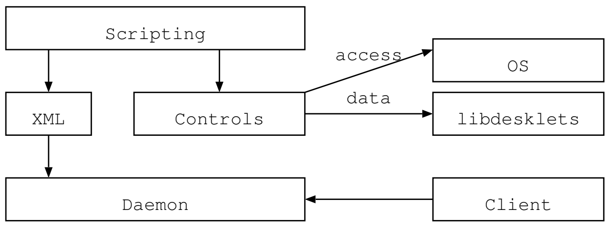


Figure 2: Components of the Framework

- A daemon is running in the background as a server providing a socket for communication. Clients connect to the socket to give instructions to the daemon, e.g. to have it open an applet.

  The daemon is responsible for loading and running the applets. Clients, e.g. a command line frontend, may thus be implemented as thin remote controls.

- The user interface of applets, i.e. the things which the user will see and be able to interact with, is defined by a simple, yet powerful XML based language.

- The XML-defined user interface can be brought to life with inline scripts, i.e. Python[2] scripts which are directly written into the XML code or imported there to be executed in a secure sandbox.

- Sandboxed scripting code may use controls to access otherwise unreachable data. Controls can, for instance, be used for reading hardware information. Basically they provide a controlled way to access data outside the sandbox. *libdesklets* is a library which can be used to ease their implementation.

3 Highlights

3.1 Technical Information

- The core is written in an efficient mixture of Python and a bit of C. That way we can combine the maintainable code of Python with the good performance of C.

- The core currently consists of roughly 10,000 lines of code. Regarding the functionality, the code base is considerably small.

- A lot of work flows into the object-oriented architecture, which is built to be easily extensible. Developer's documentation is available in CVS.[3]

[2]A widely used object-oriented interpreted language. See `http://www.python.org`

[3]The repository where the source code is stored. See `http://cvs.gnome.org`

- Easy to learn and widely adopted open standards such as XML or Python are used for developing on the platform and make it further extensible for the future.

3.2 XML User Interface Language

Many user interfaces can be built by using a set of just three simple tags (see table 1).

`<display>`	Display window
`<label>`	Textual label supporting *Pango* markup
`<image>`	Container for graphics, such as PNG, JPEG, or SVG

Table 1: The most essential tags.

Each tag provides a set of properties for configuring the element. For instance, the properties provided by `<label>` cover

- position (`x, y, anchor, relative-to`),
- the label text and font (`value, font`),
- the color of the text (`color`).

```
 1  <?xml version="1.0" encoding="UTF-8"?>
 2
 3  <display window-flags="sticky, below">
 4
 5    <image x="0" y="0" uri="gfx/gdesklets.png"/>
 6
 7    <label x="50" value="Hello, World!"
 8           font="Serif 20" color="red"/>
 9
10  </display>
```

Figure 3: The *Hello World!* of gDesklets.

The *Hello World!* example in Figure 3 illustrates their usage.
The first line contains the strictly recommended XML header which tells the parser about the used character encoding.
Lines 3 and 10 create the display window and set flags for being visible on every virtual desktop and below all the other windows.
The tags in line 5 and 7 create an image and a text label, respectively. As you can see in line 7, coordinate properties default to 0 (here `y`) and may be omitted.

More sophisticated functionality can be added by using more tags, of which the important ones are shown in Table 2.

`<group>`	Group for grouping elements together
`<array>`	Container for arrays of the same element
`<canvas>`	SVG based canvas for drawing vector graphics
`<entry>`	Text entry field
`<frame>`	Container putting a frame around its child
`<gauge>`	Horizontal or vertical gauge
`<html>`	Entry where html can be displayed
`<plotter>`	Specialized canvas for plotting curves

Table 2: Some more important tags.

3.3 Layouting

User Interface elements can be laid out flexibly in various ways which may even be combined freely. gDesklets provides

- unit-based positioning (e.g. by pixels, centimeters, inches, . . .),
- percentual positioning,
- positioning relative to other elements,
- positioning anchors (nw, n, ne, e, se, s, sw, w, center),
- and any mixture of these.

Figure 4 demonstrates the various ways and their effects.

3.4 Inline Scripting

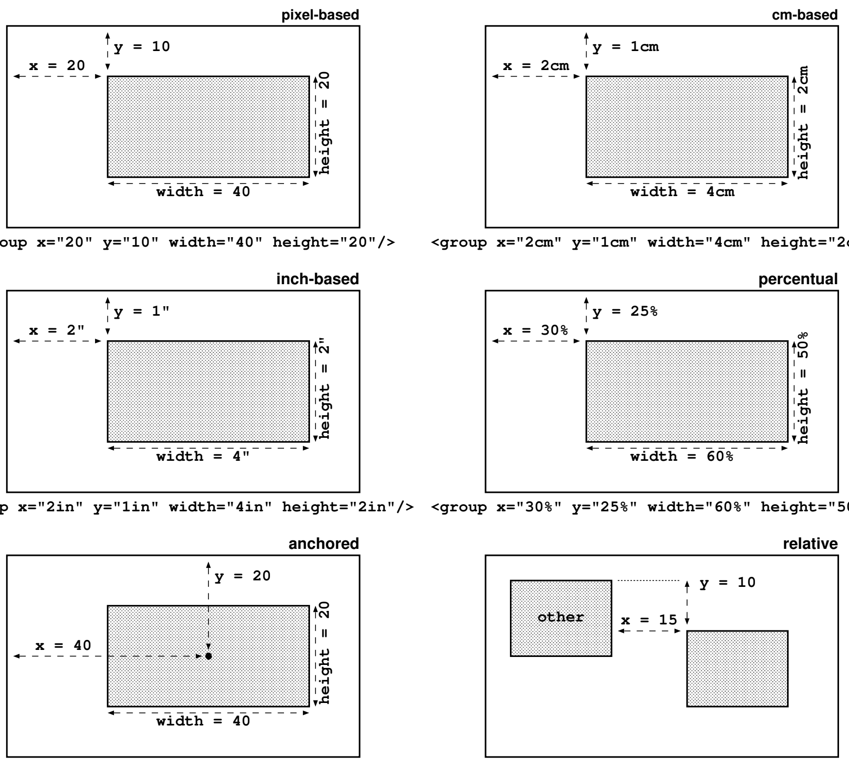
Since version 0.30, there's the possibility to put Python code into the XML files. This is called *Inline Scripting.* A simple example can be found in figure 5. Action handler properties like `on-enter, on-leave, on-click`, which are provided by most tags, and the <script> tag are the places where you can insert scripting code. The tag can also import external script files in case you want to keep your XML file clean.

The code executes in a secure sandbox disallowing privileged actions such as accessing files. Controls, which can be seen as an improvement of the former sensor concept of gDesklets, provide controlled ways to access data outside that sandbox in an object-oriented fashion through objects with readable and maybe writable properties.

Controls are basically components which get looked up by their interface identifier. They are not tied to a single desklet and may thus be shared freely. Installation and removal of these components is put under user control for safety reasons. Desklets executed off the web thus cannot install malicious code by themselves. The former sensor concept will eventually become deprecated in favor of the more flexible controls.

3.5 Remote Applets

Due to the fact that gDesklets uses *GNOME-VFS* internally for accessing files, applets may be completely located at a remote host. It doesn't matter whether



```
<group x="20" y="10" width="40" height="20"/>
<group x="2cm" y="1cm" width="4cm" height="2cm"/>
<group x="2in" y="1in" width="4in" height="2in"/>
<group x="30%" y="25%" width="60%" height="50%"/>
<group x="40" y="20" width="40" height="20"
       anchor="center"/>
<group relative-to="other, x" x="15" y="10"/>
```

Figure 4: Different ways of layouting

you access it via HTTP, FTP, SMB, or whatever your GNOME platform supports. If you combine this with the rich inline scripting capabilities and safe sandboxed execution, you can easily create secure web applications.

4 Ideas for the Future

4.1 Themes and Styles

Future versions of *gDesklets* will feature style sheets for defining the default look of elements. This will allow the implementation of *theming* in a more clean way than some desklets have it by now.

4.2 Further Work on *libdesklets*

Currently *libdesklets* provides access to system information, such as CPU, memory, swap or network devices. This will be improved and support for some non-x86 architectures will be enhanced. Since *libdesklets* is meant as a collection of libraries, we will be covering other areas as well.

```
<?xml version="1.0" encoding="UTF-8"?>

<display id="mywindow" bg-color="white">

  <label id="mylabel" value="Click Me!" font="Sans 32"
         on-enter="mywindow.bg_color='red'"
         on-leave="mywindow.bg_color='white'"
         on-click="click()"/>

  <script>

    def click():
        mylabel.value = "Thank You!"

  </script>

</display>
```

Figure 5: Example for inline scripting.

4.3 Improvements on the User Frontend

With the new *desklets-shell* it's now very easy to manage, install and uninstall desklets. We're going to work on it to make it *the* central point for managing desklets. Eventually a graphical editor for applets may be provided as a plugin for the *desklets-shell*.

4.4 More and Better Documentation

gDesklets currently has some nice documentation. However, it's not easy for beginners to get gDesklets working. Better manpages, READMEs and FAQs will help those users in the first place.

The future of rendering in GNOME

Owen Taylor <otaylor@redhat.com>

GUADEC 5

Kristiansand, Norway

June 28-30, 2004

Introduction

GNOME currently has a diverse set of rendering technologies, and of programming interfaces to use those rendering technologies. The oldest rendering technology is the core X drawing commands. These provide a very traditional 2D rendering API, including basic graphics primitives: lines, rectangles, arcs, and so forth, all without antialiasing or alpha compositing. The RENDER extension to X provides a more modern 2D set of graphics operations; in GNOME usage of RENDER is confined mostly to alpha-compositing text and graphics. On the client side, the libart library provides routines for drawing antialiased shapes, and in a more specialized domain, font rasterization is done by the FreeType library.

GNOME Applications seldom use the above technologies directly, but instead use a number of programming interfaces built on top of them. The GDK drawing interfaces used by the GTK+ toolkit generally map directly onto the Xlib commands and have similar limitations. However, for GTK+-2.0 this was slightly extended by adding support for anti-aliased text and for images with an alpha-channel. Where the RENDER extension is available, it is used to implement these capabilities. Where it isn't available, they are implemented by grabbing the destination drawable into local memory, compositing against it, and writing the result back to the X server.

The GNOME Canvas provides a retained-mode interface for drawing in GNOME. (*retained-mode* means that the application builds up a tree of graphics objects that the system redraws as necessary. Most of the other interfaces discussed here are *immediate-mode*; the application reexecutes all the drawing commands each time drawing needs to be done.) The GNOME canvas offers two drawing modes; one is non-anti-aliased and implemented using GDK. The second mode uses libart to implement anti-aliased drawing.

The gnome-print library provides yet another set of drawing interfaces for GNOME. The interfaces in gnome-print are postscript-like, but with the addition of an alpha-channel.

So, what issues need to be addressed in GNOME, moving forward? One should be apparent from the above description. The diversity of different rendering interfaces is confusing for the application programmer, and also causes problems for the implementation. Each set of rendering technologies has its own separate set of bugs and performance issues. If we can move to a unified rendering interface for screen display and printing, we solve these problems.

We might also want to provide rendering capabilities that go beyond what is offered by gnome-print. gnome-print, while it has alpha-transparency and antialiasing, is still a quite simple graphics API. It doesn't have gradients. It can't combine objects in any way other than the simplest alpha-compositing.

Cairo

Recently, a graphics library has been under development that is quite closely suited to the future rendering needs of GTK+ and GNOME. The Cairo library [Cairo] is designed to be an easy to use 2D graphics library offering a rich set of capabilities and multiple output backends.

As is the case for gnome-print, the Cairo API is closely modeled on the way that postscript works. Given a Cairo drawing context, `cairo_t *cr`, drawing a red triangle looks like:

```
cairo_set_rgb_color (cr, 1.0, 0.0, 0.0);
cairo_move_to (cr, 50.,  0.);
cairo_line_to (cr, 100., 87.);
cairo_line_to (cr, 0.,   87.);
cairo_close_path (cr);
cairo_fill (cr);
```

There are also functions to create common types of paths like rectangles and arcs and circles. But Cairo also goes beyond the simple postscript-like model in various. It makes it easy to create temporary surfaces, draw to them, then combine them back with the main surface with any of a number of different compositing modes. This gives capabilities similar to the concept of groups in PDF-1.4. It supports linear and radial gradient patterns. (Postscript Level 3 has very complicated gradient support, but this isn't found in gnome-print.)

In rough terms, the rendering capabilities of Cairo are similar to those of Java 2D, SVG, or PDF-1.4. In detail Cairo has a smaller set of capabilities than either SVG or PDF-1.4, but this makes sense; SVG and PDF-1.4 are purely declarative, so they need to cover pretty much anything an application might want to do. Cairo on the other hand, provides more opportunity for the application to build richer systems on top if needed. For screen rendering, we can even do things like render to a local buffer, perform direct operations on the pixels, then use the tweaked buffer as a source for further operations, something that would be useful, for instance, when implementing some of the SVG filter modes. An fairly implementation of SVG on top of Cairo already exists [SvgCairo].

One the backend side, Cairo currently supports a number of output targets. It has a backend that targets RENDER for X display, another that renders to a local image buffer. There's a backend that targets OpenGL that has gotten a lot of work recently, and shows that Cairo can be efficiently accelerated on modern graphics hardware. Finally there is very basic postscript backend. The backend as it exists now just renders a huge bitmap and writes that into a postscript file, which is, of course, not what you want for rendering documents. A real postscript backend needs to be quite sophisticated in figuring out what parts of the output can be represented as postscript, and what parts of the document need to be sent as bitmaps, since the Cairo rendering model goes considerably beyond what Postscript can do. While this is definitely a programming challenge, similar problems have been tackled before in such software as OpenOffice and Ghostscript, so it should definitely be doable. Another possibility is to simply write out PDF-1.4 files and let Ghostscript do the conversion to postscript; however it would be nice if a usable PS backend didn't depend on the presence of Ghostscript on the system.

GTK+ integration

GDK wraps Xlib entirely. `XDrawLine()` has the corresponding `gdk_draw_line()`, and so forth. The natural question is then whether we should do the same for Cairo. If we look at the reasoning behind the wrapping in GDK, we see that doing the same thing for Cairo doesn't make sense. By wrapping Xlib, we gain in convenience: the Xlib APIs are cumbersome to use for a number of reasons, the most obvious being that all the functions take a separate Display parameter. drawable surfaces objects rather than just numeric IDs. Unlike Xlib, programmer convenience has been one of the most important considerations when creating the Cairo APIs. That parameter was eliminated in GDK by making. Also importantly, GDK is actually a portable wrapper that can run on top of multiple rendering systems. In addition to X, GDK also runs on top of the Win32 GDI, the directfb windowing system, and several other drawing APIs. But Cairo already supports multiple backends adding another layer on top of that would add no additional benefits.

With the above considerations not being a factor, we can gain a lot by presenting the Cairo APIs directly to the application programmer. We don't need to maintain the wrapper layer; if additions are made to the Cairo APIs, they are directly available to the GNOME programmer. We don't need to maintain a separate set of documentation for our drawing library. If we have drawing libraries that we share with other non-GNOME-specific applications (for themes, for SVG rendering, or whatever), we an can simply pass the Cairo objects to them directly. There are some disadvantages to not wrapping Cairo; Cairo uses different naming conventions for types and functions then the GNOME libraries. (cairo_font_t rather than CairoFont, `cairo_font_reference()` rather than `cairo_font_ref()`). Also, because Cairo doesn't use GObject, wrapping Cairo in a language binding requires much more custom glue code than for a GNOME library. But these disadvantages don't outweigh the strong advantages of presenting Cairo directly to the application programmer.

Since we aren't wrapping individual cairo functions, the amount of API that we need to add to GDK is actually quite limited. The only function that is actually required is:

```
void gdk_drawable_update_cairo(GdkDrawable *drawable, cairo_t *cr);
```

This function redirects drawing for the Cairo surface to the given drawable. The implementation needs to take care of handling some of the internal complexities of GDK like double buffering and 32-bit coordinate emulation, but this is hidden from the user. So, an expose handler that uses Cairo is quite simple.

```
void
my_widget_expose (GtkWidget      *widget,
                  GdkEventExpose *event)
{
  cairo_t *cr = cairo_create ();
  gdk_drawable_update_cairo (event->window, cr);

  cairo_set_rgb_color (cr, 1.0, 1.0, 0);
  cairo_rectangle (widget->allocation.x, widget->allocation.y,
                   widget->allocation.width, widget->allocation.height);
  cairo_fill (cr);

  cairo_destroy (cr);
}
```

But since virtually every expose handler performs these same operations, it makes sense to provide a default expose handler that handles the creation of the Cairo context and calls a `paint` handler with that additional argument.

```
void
my_widget_paint (GtkWidget      *widget,
                 GdkEventExpose *event,
                 cairo_t         cr)
{
  cairo_set_rgb_color (cr, 1.0, 1.0, 0);
  cairo_rectangle (widget->allocation.x, widget->allocation.y,
                   widget->allocation.width, widget->allocation.height);
  cairo_fill (cr);
}
```

That's really all there is with rendering integration with GDK and GTK+. But to the ability to do rendering with an alpha-channel, we'd like to add another capability: having windows that actually have an alpha channel for a background. This is implementable with the DAMAGE and COMPOSITE extensions to X now under development, and the GDK interface is quite trivial:

```
GdkVisual *gdk_screen_get_rgba_visual(GdkScreen *screen);
GdkColormap *gdk_screen_get_rgba_colormap(GdkScreen *screen);
```

These functions gets the best available visual and colormap with an alpha channel, similar to the existing `gdk_screen_get_rgb_visual()` and `gdk_screen_get_rgb_colormap()`.

There's one further rendering trick that the COMPOSITE extension would allow us to play. Currently some GTK+ widgets have their own X windows, other GTK+ widgets draw directly into their parent's window. This is a useful compromise because not having a separate window is slightly more efficient and allows better rendering (in particular proper blending with the parent widget), while having a separate window allows taking advantage of X's facilities for scrolling and clipping. However, the mixture causes problems for controlling Z order: widgets with a window will inevitably draw above widgets without a widget. The COMPOSITE extension allows for fixing this; we could add a function:

```
void gdk_window_set_parent_draw(gboolean parent_draw);
```

When this flag is turned on, changes to a child window do not have any immediate affect on the screen. Instead, they just cause the corresponding areas to be added to the parent's invalid region. The application or widget is responsible for drawing the child windows onto the parent widget, and can properly interleave child widgets with and without child windows into the proper Z order. The question here is whether this facility is useful without being available everywhere; it's not possible to implement a fallback implementation that works on older X servers. [1]

Text

One thing that was glossed over in the preceding section is how text drawing works. This is an area where what GTK+ applications use will be significantly different from the raw API, because GTK+ is based on Pango, which provides a much richer than the simple text API that Cairo provides itself. The simplest use of Pango in a Cairo program will look like:

```
PangoLayout *layout = pango_cairo_create_layout (cr);
pango_layout_set_text (layout, "Hello world");
pango_cairo_show_layout (cr);
g_object_unref (layout);
```

The `pango_cairo_create_layout()` here is a convenience function that looks like:

```
PangoLayout *
pango_cairo_create_layout (cairo_t *cr)
{
  PangoFontMap *font_map = pango_cairo_get_default_font_map ();
  PangoContext *context = pango_cairo_font_map_create_context (font_map);
  PangoLayout *layout = pango_layout_new (context);

  pango_cairo_context_update (context, cr);
  g_object_unref (context);

  return layout;
}
```

This may look a little different than the initial expectations. You might expect a PangoContext to be created for a particular cairo_t. However, this doesn't work because Cairo context is a transient object that we create just when we are rendering, but it's useful to keep PangoLayout objects around in many cases, since layout is an expensive operation. And each PangoLayout contains a persistant pointer to a PangoContext. So, the PangoContext doesn't contain a pointer to the cairo_t; rather it just copies the information about the current transformation matrix and destination surface out of the cairo_t that is needed for doing layout.

It should be emphasized that all the dimensions and positions associated with a PangoLayout object are in user coordinates, not device coordinates. The advantage of this is considerably simplicity. We can say that lines of text always run in the X direction, and that paragraphs always lay out as an axis-aligned rectangle. In fact, even if we have Chinese writing where the lines of text run top-to-bottom on the page, the text still runs in the X direction, we just require the application to use a 90 degree rotation. But then you might wonder why the layout depends on the current transformation matrix at all. Shouldn't the positions just scale and transform exactly with the size of the font? In some cases, using layout that is independent of the current transformation is useful; this is what we want if the final output is a high resolution printer. But if we are optimizing text for display on the screen, we can produce significantly better looking output by positioning using the metrics for a particular pixel size [Taylor1].

Themes

[1]It might seem you could implement the child windows as pixmaps, but this doesn't work because we expose the fact that each GdkWindow corresponds to an X window

Now that we know how widgets interact with Cairo to render themselves, we then need to look at what gets drawn by these widgets. Theming is difficult issue because there is an inherent tension. On one hand, we want to be able have themes that can control precisely how GTK+ renders, and we want to be able to extend GTK+ with new and novel types of widgets. These considerations argue for a theming system that is very tightly tied to the way that GTK+ works. On the other hand, we want to be able to write GTK+ themes that chain to a platforms native look; the GTK-WIMP [GTK-WIMP] project has done this very effectively for Windows. And we want to be able to use the GTK+ theme system to render other widget sets; this is currently being done by OpenOffice and Mozilla. Those considerations militate for a theming system that is much more closely tied to an idea of a "standard set" of widgets. In addition there is the issue of third party widgets. It has to be possible to for libraries and applications to create new types of widgets and have them integrate into the theming system and in fact work with themes that were created without any knowledge of these new widget types.

Many of these considerations were in fact known when the current GTK+ theming system was created in 1998, and the attempt was to create a maximally flexible system. The way that the GTK+ theming system works is that there is a set of paint functions corresponding to different basic widget system components: flat and beveled boxes, check-button indicators, arrows, notebook tabs, and so forth. A theme engine provides implementations of each of these functions, and when the function is called receives not only the destination drawable and information about where to draw the component, but also extra information: a detail string, which is an unspecified string giving extra information about the particular usage of the component and a pointer to the widget itself. The idea is that by providing basic implementations of the component functions, the theme engine can minimally render any widget, but it can also use the detail string and even the widget pointer to special case and provide improved rendering for particular widgets. The theme engine along with generic and theme-engine specific options are bound to particular widgets using the gtkrc file language [Taylor2].

While the current theming system has been successful in the sense that people have generally managed to work with it and get the appearance that they desire, many deficiencies have been found. The set of detail strings used is unspecified and in practice themes engines *do* need to special-case a consider number of details to render GTK+ widgets correctly, so creating a theme engine is an exercise in cut-and-paste and trial-and-error. Typically theme engines also end up referencing the widget pointers that are passed in; while these pointers are in theory allowed to be null and theme engines are supposed to check this, in practice an attempt to pass in null pointers when rendering controls that aren't GTK+ widgets will crash most theme engines. The tight binding of theme engine rendering to particular types of widgets also causes problems when creating custom widget types. It's very difficult to create a custom widget that appears like a `GtkEntry` but isn't actually one. Even the simplest case of deriving a new widget from an existing class can break themes. And finally, there is no conception of layout in the theme system; the way that the components of a widget are fit together to create the widget has to be figured out by studying the GTK+ sources. This causes particular problems for people using the GTK+ system to render non-GTK+ widgets, because they have to duplicate considerable amounts of layout code from GTK+.

While designing a new theme system for GTK+ is beyond the scope of this paper, we can lay out some general principles for how it should work:

- It should be implemented without reference to widget specifics so that it can be used by anybody who needs to render a control that matches the GTK+ look. In fact, it may be desirable to implement it with minimal dependencies on the GTK+ stack to further widen the set of possible consumers. Making Cairo the rendering interface would bypass a lot of potential problems with finding a common ground for rendering.
- It should be multi-layered. While the idea in the current GTK+ theme system of standard components that can be recycled to create custom widgets is likely useful, it's not sufficient. If we add an extra layer on top of that that has an idea of complete controls and of layout, then we make the theme system both more flexible for theme creators and more easily usable for theme consumers.
- As much as possible should be declarative; config files should be used instead of code. However the ability to chain to native code needs to be preserved to do things like a native theme on Windows.
- Careful specification is essential. If there is only one provider: one theme engine, or only one consumer: one set of controls, then implicit specification by implementation may work, but we are in a case where we have multiple providers and multiple providers and multiple consumers, so a formal specification is crucial.

- For declarative parts, it should use standard file formats such as XML and possibly CSS instead of inventing custom syntax.

It should be possible to compatibly introduce the theme system sketched above into GTK+ by implementing it as a theme engine; parts of GTK+ and 3rd party widgets could then be gradually transitioned over to using the new system directly.

Printing

The existence of PS and PDF backends for Cairo clearly goes a long ways toward providing consistent interfaces for rendering to screen and printing, but it's not a complete solution for application printing. We need a way to put a dialog for the user to select a printer and choose options for the printer. The application needs to be able to get basic information about the chosen printer; information like paper size and whether it's a color device or monochrome device. And finally the application needs to be able to create a Cairo context that spools output to this printer taking the selected options into account. This type of functionality is currently provided by libgnomeprint and libgnomeprintui.

The natural place for this functionality to live is in GTK+. This is functionality that virtually all applications need. It's functionality that needs to be done significantly different depending on the platform; on Windows, for example, we want to see the printers configured on the system, to print to them with a GDI backend for Cairo, and possibly to use the native print dialogs. For Linux, we'd want to have tight integration with CUPS. We might also have a lpr-based backend for legacy Unix systems. And finally its not a huge amount of code once we have the Cairo backend. libgnomeprintui is only about 15,000 lines of code.

The API here should be straightforward. We'd have a print dialog that would would work similarly to `GtkFileChooser`; the application could then retrieve a `GtkPrintContext` object reflecting the printer and options that the user selected in the dialog. The application could retrieve information about the printer from the `GtkPrintContext` and create a cairo_t that renders to the printer.

Conclusion

We've seen that currently rendering in GNOME is done with a ad-hoc collection of different interfaces and technologies. Cairo offers an appealing way to both unify on a single rendering interface, and to improve the rendering capabilities provided to GNOME applications. Moving to Cairo provides us the opportunity to revisit such areas as printing and themes, solve some of the long-outstanding issues, and make sure that these capabilities are provided at the right level in the platform stack.

References

[Cairo] *Cairo vector graphics library [http://cairographics.org/].*

[GTK-WIMP] *GTK-WIMP [http://gtk-wimp.sourceforge.net/].*

[SvgCairo] *libsvg-cairo [http://cairographics.org/libsvg-cairo].*

[Taylor1] Taylor, Owen. *Rendering good looking text with resolution independent layout [http://people.redhat.com/otaylor/grid-fitting/].*

[Taylor2] Taylor, Owen. *The GTK+ Theme Architecture, version 2 [http://www.gtk.org/~otaylor/gtk/2.0/theme-engines.html].*

Automatic Verification Techniques to Improve the Open Source Software Quality

Paolo Maggi and Davide Pozza
Dip. di Automatica e Informatica - Politecnico di Torino
Corso Duca degli Abruzzi 24, I-10129 Torino, ITALY
Email:{paolo.maggi,davide.pozza}@polito.it

Abstract— **System verification techniques have a great importance in the design and implementation of reliable information systems. Historically, peer reviewing and testing were the major verification techniques used in the development of open source software systems. However, due to the increasing complexity of such systems, they are no more sufficient in providing an adequate level of quality. In this paper, we will show how the adoption of suitable automatic verification techniques can help to improve the quality of software systems. In particular, we will first show how to use model checking techniques for validating part of the D-BUS specification. Then, we will present some tools for the automatic verification of C code that can be used by developers and security auditors to detect programming errors that can lead to security flaws like buffer overruns, format string attacks, etc.**

I. Introduction

In our every day life we increasingly rely on information systems whose complexity, and hence whose vulnerability to errors, is constantly growing. Errors in safety-critical systems (nuclear power plants, biomedical instruments, flight control systems, etc.) are unacceptable and may lead to catastrophic consequences, but also errors in other kinds of application (our preferred word processor, the database server of our bank, etc.) can have a great impact on our life. For these reasons, reliability, correctness and security of software components are becoming more and more important.

System verification techniques can have a great impact in the design and implementation of reliable information systems since they can be used to determine the correctness of specification, design and implementation. Moreover, they can be used to verify whether the system satisfies a predetermined set of properties.

Historically, peer reviewing and dynamic testing (design tests, system tests, field tests, etc.) were the major verification techniques used in the development of software systems and, in particular, of the free and open source ones. However, due to the increasing complexity and magnitude of such systems, they are no more sufficient to provide an adequate level of quality. Consequently, the use of additional verification techniques should be taken into consideration, especially when they can be, at least partially, automated.

Apart from peer-reviewing and testing the most important verification techniques are the so-called formal verification techniques. These ones work on models (eventually extracted from the implementation) and, in the ideal cases, aim to the mathematical proof of the correctness of a system.

In this paper, we will give an overview on the available automatic verification and error detection techniques focusing on the ones that can be used for improving the reliability and the security of open source software systems. In particular, in section II, we will show how to use model checking (an automated formal verification technique) for validating system specifications, taking part of the D-BUS specification [1] as a case study. In section III, we will describe some tools, that work on source code, that can help to verify the adherence of implementations to their specifications and to detect the most common classes of programming errors that can lead to security flaws. The tools presented in that section focus on the C language, being this one the most widely adopted languages in the free and open source software community. In the same section we will also propose a possible implementation methodology that makes use of the described tools. Section IV concludes. Appendix I briefly describes some popular classes of implementation security vulnerabilities.

This work was partially supported by the Center for Multimedia Radio Communications (CERCOM) of Politecnico di Torino.

II. Using formal verification in the early phases of the development cycle

Although verification techniques are often seen as a way for improving the quality of already developed code, they can be successfully used also in the early phases of the development cycle. In particular, the use of formal verification techniques during the analysis and design phases has proved to be really important for the development of reliable and secure information systems. These techniques work on models that describe the behavior of the system in a mathematical precise and unambiguous manner and aim to the mathematical proof of the correctness of the system. Given a model of a system and a set of correctness requirements written in the form of formal requirements specifications that represent the desirable behavior of the system, there are two different ways we can verify if the possible behavior *agrees with* the desired behavior, i.e. if the model satisfies the correctness requirements. The first way is to use the so-called *theorem proving* techniques that consist in developing a mathematical proof of correctness in a similar way one can prove, for example, the Pythagorean theorem. Even if some automated *theorem provers* have been developed the big disadvantage of

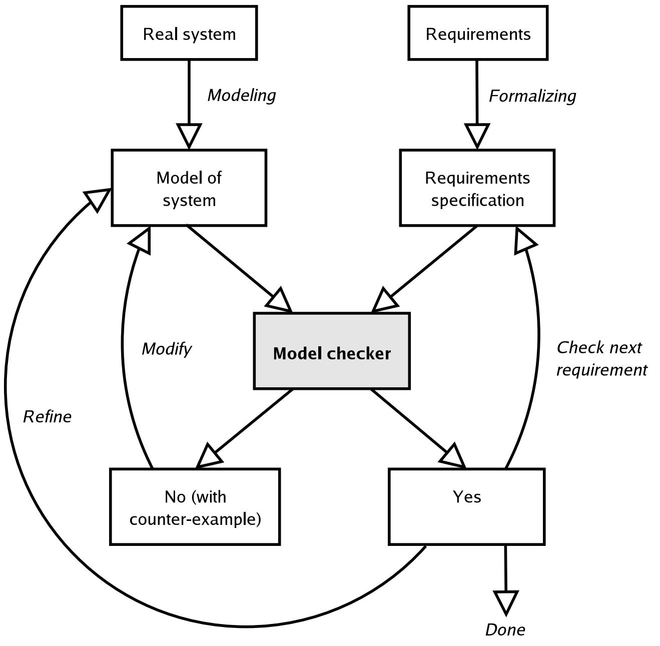


Fig. 1. The methodology normally followed when using model checking.

this approach is that it is tedious, labor-intensive and requires a high degree of expertise. The second possible approach, called *model checking*, normally requires a lower involvement of the user in the verification process and hence it can be usefully adopted also by people with a lower degree of expertise.

Model checking uses algorithms, executed by a computer, to verify the correctness of systems without any user interaction. These algorithms typically perform an *exhaustive state space search* of the model of the system, i.e., for each state reached by the model, it is checked if the desired properties are satisfied. The methodology normally followed when using model checking is shown in Fig.1.

An important feature of model checking is that, if the model of the system does not satisfy the requirements, the model checker provides a counter-example showing under which circumstances the requirements are not satisfied.

To show how model checking can be used for improving the quality of free and open source software, we have tried to analyze part of the D-BUS specification using the open source model checker SPIN.

SPIN[1] [2], [3] is one of the most efficient model checkers available to date. The input language of SPIN is called PROMELA (PROcess MEta LAnguage) and provides a simple, intuitive, program-like notation for modeling concurrent programs and systems. While the PROMELA syntax is mainly based on the C language, the control flow primitives are related to the guarded selection statements used in languages such as Ada [4] and Orca [5].

Given a system specification in PROMELA, the model checker SPIN translates each process definition into a finite state automaton. Then it can either perform a random simulation of the global system behavior, or generate a C program that performs an exhaustive verification of the model using reachability analysis. Since the size of the state space can grow exponentially with the sizes of the component processes, SPIN uses a number of complexity management techniques to contain its explosion. Among these, we can cite partial order reduction, state-vector compression and bit-state hashing, also known as *supertrace*. The latter technique can be used to perform a partial verification and to evaluate its coverage when exhaustive verification is impossible.

During simulation and verification SPIN automatically checks for the absence of deadlocks and assertion violations. It can also be used to check for non-progress cycles and to verify correctness properties expressed in the form of Linear Temporal Logic (LTL) formulas. Once a verification error has been found, SPIN can generate a trace of the system states that leads to the error.

The part of the D-BUS specification we have analyzed is the "Authentication Protocol". First of all we have written a PROMELA model of the protocol based on the "Authentication state diagrams" presented in the specification. Even if the specification claims that "this is probably the most robust way to implement the protocol", we have found that the reported state diagrams are seriously underspecified and hence we have been obliged to make some guess work for writing our model. This is a quite frequent problem while developing formal models starting from informal or semi-formal (like in this case) specifications. Indeed, an important side effect of the use of model-based verification techniques is that, prior to any form of verification, the accurate modeling of systems normally leads to discover incompleteness, ambiguities and inconsistencies in the informal system specification. These ones are problems that can lead to incompatible and non-interoperable implementations. Without using formal verification techniques, they are normally discovered only in later phases of the development cycle, when the impact on the overall costs can be really great.

The developed model is really simple, it consists of three processes: a process for the client, one for the server and an `init` process that instantiates the client and the server processes. The client and the server communicate with each other using a global synchronized channel. In order to evaluate the robustness of the system, we have also developed a second model containing a special process acting as an error generator which removes messages from the channel and/or adds new unknown ones. We have verified both models searching for deadlocks and non-progress cycles, but we have not found any particular problem.

The D-BUS specification also describes an ad-hoc authentication mechanism called `DBUS_COOKIE_SHA1`. This mechanism has been designed to "establish that a client has the ability to read a private file owned by the user being authenticated". Even if we have not explicitly analyzed the `DBUS_COOKIE_SHA1` mechanism, it is worth noting that, despite their apparent simplicity, this kind of mechanisms, based on cryptographic protocols, have revealed themselves to be very error prone, especially because of the difficulty generally found by their designers in foreseeing all the possible attacks and all the possible behaviors of various parallel protocol sessions. For this reason, when a new cryptographic

[1]It is available at http://www.spinroot.com

protocol is designed, it is always advisable to try to analyze it using formal verification techniques. In the last years, researchers have investigated both the theorem proving and the model checking approaches to the verification of security properties of cryptographic protocols and some of the tools they have developed are freely available. For example, some model checkers specifically designed to analyze cryptographic protocols have been developed (e.g. [6], [7]), and various approaches using general purpose model checkers have been described too. In particular, a possible approach that makes use of the SPIN model checker is presented in [8].

III. Source Code Verification

When an high confidence about the architectural correctness of the system has been reached, source code analysis activities can take place. Results obtained during the architectural analysis of a system should provide a list of potential risks for the system. This list must be used as a reference during implementation analysis to better focus on the most risky parts of code.

The implementation analysis is twofold. First it is necessary to verify whether the implementation is effectively coherent to the system design. Then the errors introduced during the implementation phase must be fixed. It is worth noting that some errors, such as buffer overflows, are implementation specific and hence can be discovered only during the source code analysis.

Checking the coherence between design and implementation is an hard task that usually consists in proving by hand that everything is implemented as specified during the design phase. A different approach to the problem is to use code generation tools which, starting from a model whose correctness can be formally verified, automatically generate an implementation whose behavior is guaranteed to be the same of the model. Good results on automatic code generation have been obtained in the field of cryptographic protocols as documented in [9].

To help the developers to perform the coherence analysis above described, some tools have been developed. These tools can also help developers to detect violations of temporal safety properties that can lead to various kinds of security problems.

MAGIC[2] (Modular Analysis of proGrams In C) [12][13] is a tool that automatically verifies C programs (both sequential and concurrent ones) against finite state machine specifications. In practice, MAGIC is able to verify whether the implementation of a system is safely abstracted away by its specification. The tool takes in input a system specification expressed as a set of Labeled Transition Systems (LTS). Then the tool builds an implementation LTS for each implemented procedure and verifies the conformance between LTSs via weak simulation. Note that a procedure may in turn invoke other procedures which are themselves specified as LTSs. If there is no conformance, a counterexample is given. MAGIC can be used throughout all the development cycle since it allows the composition of specifications (i.e. specification LTSs can be plugged in for missing components).

BLAST[3] (the Berkeley Lazy Abstraction Software verification Tool) [10][11] is a verification tool for checking safety properties concerning the sequence of program "events" in C code. An example of safety property is: "if a resource is locked by the `lock()` function, the resource cannot be locked again unless the resource has been unlocked using the `unlock()` function". The tool takes in input a C program and a safety monitor written in C. The program and the monitor are compiled into a single program that contains a special error state that can be reached only if the program does not satisfy the safety property. So the tool checks the abstracted model of the resulting program and, in the case it does not fail to terminate, returns either an error trace or a proof of correctness.

MOPS[4] (MOdel Checking Programs for Security Property) [16] uses a formal approach to find security bugs and to verify their absence. This tool works by detecting violations of ordering constraints, i.e. it verifies that in all the possible program paths the order of the security-relevant function calls is respected, otherwise it reports an error trace. The most important features of this tool are that it is sound in verifying the absence of certain classes of vulnerabilities and it is fully inter-procedural. Some property specifications are provided with the tool itself. Though, it is also possible to specify new properties using an ad-hoc language for describing FSAs (Finite State Automaton). Some examples of property are: "`chroot()` must always be immediately followed by `chdir()`", "`stat(f)` must not be followed by `open(f)`", "`execl()` must not be called in privileged state", etc.

The problem of avoiding and detecting vulnerabilities introduced in the coding phase can be approached in different ways. A possible approach is the use of tools that can help developers to detect the code points where vulnerabilities can be introduced. Tools like ITS4[5] [14][15], Rats[6], and Flawfinder[7] belongs to this class of tools. They are based on simple static analysis techniques and can be seen as advanced "`grep`" commands which make use of heuristic algorithms to reduce the number of false positives (warnings that do not correspond to real bugs) and to associate to the reported warnings a risk level. Such tools use a database of known potentially insecure functions, and scan source code for potential security issues such as buffer overflows, format string bugs, race conditions, insecure temporary file usage and weak randomness (see appendix I). ITS4 and Flawfinder are able to analyze programs written in C and C++, while Rats is also able to analyze Perl, PHP and Python code.

Results provided by these tools consist in a report which lists all the lines of code containing potentially insecure functions and, for each item of the list, in a description of the possible problem and, eventually, in a suggested solution. These reports are very useful for programmers, because they can help them to become more conscious of the risks involved with the use of certain functions.

Another important feature of these tools is that they can show their results in different ways, for example sorting them

[2]It is available at http://www-2.cs.cmu.edu/∼chaki/magic/

[3]It is available at http://www-cad.eecs.berkeley.edu/∼rupak/blast/

[4]It is available at http://www.cs.berkeley.edu/∼daw/mops/

[5]It is available at http://www.cigital.com/ITS4/

[6]It is available at http://www.securesw.com/

[7]It is available at http://www.dwheeler.com/flawfinder/

by risk level, thus increasing programmers' attention toward those errors that can be more dangerous. It is important to know that these tools do not report all the code points where a vulnerability can be introduced, but only the ones involving functions included in their database.

It can be advisable to use ITS4, Flawfinder or Rats to make a focused code review after every function or little piece of code (may be 100 lines of code) has been completed. In this way programmers can check if the code they have just developed contains or introduces bugs.

A good practice for developing secure code is to write a function and then check its correctness, verifying that it strongly satisfies all the assumptions that have been made.

It is worth noting that functions which are considered insecure (e.g. `strcpy()`) can become secure when used wrapped inside the right control logic, whereas more secure functions (e.g. `strncpy`) may introduce bugs too. This is the reason ITS4, Flawfinder and Rats provide only warnings on points where a bug can be introduced.

Another interesting tool is Splint[8] [17][18][19]. It is a static analysis tool that does not look only for security vulnerabilities in C code, but also for other kinds of coding mistakes and for bad coding styles. Its scanning process is much more sophisticated than the one of tools like ITS4, since it also looks for issues such as null pointer dereferences, memory leaks, ignored return values, empty bodies of `if`, `while` or `for` statements, and other stylistic issues (like having comment symbols within comments). Splint, for example, does not detect only buffer overflows caused by functions, but also those involving buffer accesses by means of indexes and pointers.

Even if Splint is a sophisticated tool it has a big limitation: it is neither sound, nor complete (i.e. it produces both false positive and false negative results). This is due to the particular analysis techniques it uses. In fact, it performs an intra-procedural data flow analysis and uses heuristic mechanisms to understand some common loop idioms in order to have a flow sensitive analysis. Whenever heuristics fail, the analysis is flow insensitive.

The main drawback in the usage of Splint is that it can be tedious to use when the user is only interested in some specific kind of vulnerabilities, since, by default, the tool reports too much warnings. Looking for the right flags to suppress the non interesting warnings can take a lot of time, so that the use of the "`grep`" command can be advantageous with respect to the built-in flags of Splint. For example, in order to look for buffer overflow write bugs only, the command can be: `splint -weak +boundswrite sourcefile.c | grep "out-of-bounds" -A8`. A disadvantage of Splint in comparison with ITS4, Rats and Flawfinder is that it does not provide a way for ordering results (for example by risk level).

Splint allows the use of code annotations, which are stylized comments, in order to suppress spurious warnings and to perform a more accurate analysis. Annotations are used to provide extra information about assumptions on: variables, function parameters, return values, structure fields and type

```
 1:#define MAXLEN 25
 2:
 3:static void function1(char *buffer)
 4:/*@requires maxSet(buffer)>=9@*/
 5:/*@ensures maxRead(buffer)==9@*/
 6:{
 7:  strcpy(buffer, "function1");
 8:}
 9:
10:static function2(char *buffer)
11:/*@requires maxSet(buffer)>=(maxRead(buffer)+9)@*/
12:/*@ensures maxRead(buffer)==(maxRead(buffer)+9)@*/
13:{
14:  strcat(buffer, "function2");
15:}
16:
17:static void function3(char *buffer)
18:/*@requires maxSet(buffer)>=(maxRead(buffer)+9)@*/
19:/*@ensures maxRead(buffer)==(maxRead(buffer)+9)@*/
20:{
21:  strcat(buffer, "function3");
22:}
23:
24:int main() {
25:  char buffer[MAXLEN];
26:
27:  function1(buffer);
28:  function2(buffer);
29:  function3(buffer);
30:  return 1;
31:}
```

Fig. 2. Code example with Splint annotations.

```
example.c: (in function main) example.c:29:2:
Likely out-of-bounds store:
    function3(buffer)
    Unable to resolve constraint:
    requires 24 >= 27
    needed to satisfy precondition:
    requires maxSet(buffer @ example.c:29:12) >=
     maxRead(buffer @ example.c:29:12) + 9
    derived from function3 precondition:
    requires maxSet(<parameter 1>) >=
     maxRead(<parameter 1>) + 9
    A memory write may write to an address
    beyond the allocated buffer. (Use
    -likely-boundswrite to inhibit warning)
```

Fig. 3. Splint output for the annotated code.

definitions. Fig. 2 shows a code example where annotations suppress spurious warnings about potential buffer overflows and enable identification of the exact code point where the bug is inserted. The output of Splint, for the same code without annotations, provides warnings on lines 7, 14 and 21. This is because Splint does not have any information about the characteristics of function parameters when it analyzes procedure bodies. When the code is annotated, Splint can verify whether function parameters ensure what is implied by annotations, so it can perform a more accurate analysis raising warnings exactly where bugs are introduced. Fig. 3 shows the Splint output for the annotated code of Fig. 2. Note that Splint does not report only the line where it thinks to have found a bug, but it also shows which are the constraints it is not able to resolve, so the user can have useful information to verify whether the warning really corresponds to a bug.

It is highly recommendable to annotate every function as soon as it has been completed (even if this is a really tiring activity). In this way all assumptions about functions will be explicitly verifiable by Splint. Moreover when all functions will be integrated together, a better analysis will be performed

[8]It is available at http://www.splint.org/

so as to obtain a potentially lower number of false positive results.

When running ITS4 and Rats on code of Fig. 2, no warnings are reported, because these tools consider `strcpy` and `strcat` as low risk functions, when the second parameter is a constant string. From a security point of view this consideration is correct, since these buffer overflows cannot be exploited by attackers. When ITS4 is ran, it is possible to disable heuristics that can lead to these kinds of false negatives using the `-H` option. This option disables the use of heuristics, so that the "severity" of functions like `strcpy(dst, "fixed")` is not reduced from level 2 to 0. In this way, using the default setting (2) for the severity cutoff, the tool reports them as potential flaws. Another way to avoid these potential false negatives is to set the severity cutoff to level 0, by means of the `-c` option. Low severity vulnerabilities can be reported by Rats, when its warning level is set to level 3, by means of the `-w` option. By default, when Flawfinder analyzes the code of Fig. 2, it reports warnings on lines 7, 14 and 21, but it is also possible to modify its warning cutoff using the `-m` option.

In conclusion, Splint can be used to find a lot of code problems starting from the early developing phases to the last ones. It is advisable to first use Splint for looking for all the potential problems it is able to address and then to use ITS4, Flawfinder or Rats. To instruct these tools to ignore the lines containing false positive warnings, special comments (like `// ITS4: ignore`) can be added to the code.

The use of Splint should be an iterative process, since it produces warnings that can lead to either changes in the code or annotations. Therefore Splint must be ran until no (relevant) warning is reported. Since Splint is very fast to perform analysis, the re-run process is not burdensome.

While programmers may use ITS4, Rats and Flawfinder without modifying their warning cutoffs to verify code during the early coding phase, it is highly recommended to make at least one auditing of the full code, setting the tool options in order to provide the highest number of warnings. This can lead not only to more false positives, but also to more discovered bugs.

After the coding of each function is completed and the auditing for a specific set of errors is done, it is advisable to immediately annotate the function. Then, it is possible to perform a modular verification of functions, analyzing them with MAGIC, to check that implementations agree with their specifications.

When all components have been annotated and have been integrated together a new auditing phase should be performed using Splint. At the end, MAGIC, BLAST and MOPS can be used to find temporal logic errors and to verify their absence over the whole program. We suggest to use MAGIC and BLAST only after a good auditing phase has taken place, since programming errors, like out of bounds indexing of buffers, can cause misunderstandings of the program behavior.

IV. Conclusions

In this paper we have presented some verification techniques which can be used to design and implement more reliable, robust and secure software systems. Moreover we have given some guidelines on when and how to use these verification techniques in order to obtain the best results, since the whole verification process must be well organized, well structured and well planned.

It was experimentally demonstrated that catching bugs as soon as possible is better since the cost of removing bugs increases dramatically as late as they are discovered. Thus it is advisable to spent the right time to use the tools and the techniques presented in this paper since it is a good investment.

It is worth noting that no tools will discover all bugs and will eliminate all security risks. However a large number of bugs can be discovered and avoided using these tools, hence they should increasingly become part of the development process.

At present using these tools still requires heavy human interaction and a non-negligible amount of time, but it should be considered that these are quite new technologies still being improved. In the future we think that such kind of tools will be more automatic and will produce more accurate results.

Appendix I
Popular Classes of Implementation Security Vulnerabilities

In this section we give a brief overview of the most common classes of implementation security vulnerabilities.

Buffer overflow involves buffers, such as strings and arrays. If poor on no checks are performed on buffer operations (both writing and reading), data can be written or read outside the buffer bounds, hence a buffer overflow occurs. Writing data outside the buffer bounds, an attacker can insert and execute new code. Even partial overwriting of variables can lead to some risks, because the program behavior can be modified. If an attacker is able to read outside the buffer bounds, it could gain access to sensitive data.

Format strings define how an output will be formatted. If an attacker has the ability to dictate what format strings will be used in a formatting command, such as the one of `printf()` family functions, consequences similar to the ones due to buffer overflows are possible.

Race conditions can happen when more than one program, thread or process are competing for the same resource. A malicious program could grab resources before any other entity can get them.

Insecure file usage is a race condition problem. A malicious program can gain control of and manipulate files through a race condition.

Weak sources of random numbers can have a great impact on security since there are several security mechanisms, such as cryptographic functions, that rely on random numbers. Without a good source of random numbers, cryptography will be weaker. For example, if an attacker can observe which random numbers have just been generated it could be able to guess the next generated number. If this number is used as a cryptographic key the attacker could be able to decrypt all the messages encrypted with the guessed key.

References

[1] A. Carlsson, A. Larsson, H. Pennington, “D-BUS Specification”, version 0.8, available at http://freedesktop.org/Software/dbus/doc/dbus-specification.html

[2] G.J. Holzmann, *Design and Validation of Computer Protocols*, Prentice Hall, 1991.

[3] G.J. Holzmann, “The Model Checker SPIN”, *IEEE Trans. on Software Engineering*, Vol. SE-23, No. 5 (May 1997), pp. 279-295.

[4] I.C. Pyle, *The Ada Programming Language*, Prentice Hall, 1981.

[5] H. Ball, *Programming Distributed Systems*, Prentice Hall, 1990.

[6] E.M. Clarke, S. Jha, W. Marrero, “Verifying security protocols with Brutus”. *ACM Trans. on Software Engineering and Methodology*, 9, 2000, pp. 443–487.

[7] L. Durante, R. Sisto, A. Valenzano, “Automatic testing equivalence verification of SPI calculus specifications”, *ACM Trans. on Software Engineering and Methodology*, Vol. 12, No. 2 (April 2003), pp. 223–284.

[8] P. Maggi, R. Sisto, “Using SPIN to Verify Security Protocols” *Proc. 9th Int. SPIN Workshop on Model Checking of Software*, LNCS 2318, Grenoble, France, April 2002, pp. 187-204.

[9] D. Pozza, R. Sisto and L. Durante, “Spi2Java: Automatic Cryptographic Protocol Java Code Generation from spi calculus”, *Advanced Information and Networking Applications*, 2004.

[10] T. A. Henzinger, R. Jhala, R. Majumdar and G. Sutre, “Software Verification with Blast”, *Proceedings of the 10th SPIN Workshop on Model Checking Software*, Lecture Notes in Computer Science 2648, Springer-Verlag, pp. 235–239, 2003.

[11] T. A. Henzinger, R. Jhala, R. Majumdar and G. Sutre, “Lazy Abstraction”, *ACM SIGPLAN-SIGACT Conference on Principles of Programming Languages*, pp. 58–70, 2002.

[12] S. Chaki, E. Clarke, A. Groce, S. Jha and H. Veith, “Modular Verification of Software Components in C”, *Transactions on Software Engineering*, to appear.

[13] S. Chaki, E. Clarke, A. Groce, J. Ouaknine, O. Strichman and K. Yorav, “Efficient Verification of Sequential and Concurrent C Programs”, *Formal Methods in System Design*, to appear.

[14] J. Viega, J.T. Bloch, T. Kohno and G. McGraw, “ITS4: A static vulnerability scanner for C and C++ code”, *Annual Computer Security Applications Conference*, Dec. 2000.

[15] J. Viega, J.T. Bloch, T. Kohno and G. McGraw, “Token-Based Scanning of Source Code for Security Problems”, *ACM Transactions on Information and System Security*, Vol. 5, No. 3, pp.238–261, August 2002.

[16] H. Chen and D. Wagner, “MOPS: an Infrastructure for Examining Security Properties of Software”, *ACM Conference on Computer and Communications Security*, 2002.

[17] D. Evans and D. Larochelle, “Improving Security Using Extensible Lightweight Static Analysis”, *IEEE Software*, 2002.

[18] D. Evans and D. Larochelle, “Statically Detecting Likely Buffer Overflow Vulnerabilities”, *USENIX Security Symposium*, 2001.

[19] D. Evans, “Static Detection of Dynamic Memory Errors”, *SIGPLAN Conference on Programming Language Design and Implementation*, 1996.

Paolo Maggi graduated in Computer Engineering in 1998, and received a Ph.D degree in Computer Engineering in 2002, both from Politecnico di Torino, Torino, Italy. His current research interests include distributed systems design, mobile agents security and formal methods for cryptographic protocols.

Davide Pozza received his MS degree in Computer Engineering at Politecnico di Torino, Torino, Italy in 2002. He is now a Ph.D student in the Department of Computer and Control Engineering at Politecnico di Torino. His main research interests are in the area of software security, in particular he is working on techniques for detecting and preventing security breaches at the source code level.

Writing a GIMP Plug-in

David Neary

May 15, 2004

1 Introduction

In this paper, I will present the basics of a GIMP plug-in, including the required elements of a plug-in, an introduction to the libgimp API, accessing image data efficiently, and using the PDB to make our plug-in available to scripters.

Many people are intimidated by the GIMP, and imagine that writing a plug-in would be a difficult task. The objective of this presentation is to show that there are very few difficulties to writing a plug-in in C, once we know the basics.

During the presentation I will implement a plug-in with a trivial algorithm. I will then illustrate the use of the plug-in, both interractively and via a script.

2 Architecture

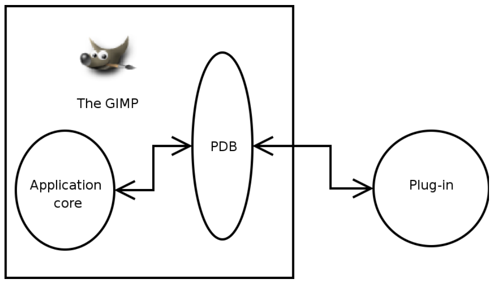


Figure 1: The Procedural Database

The GIMP's scripting interface is based around the Procedural Database. At start-up, the GIMP looks in a number of predefined places, and runs and queries each executable in those places.

The plug-in declares itself at this point to the PDB, including information such as its position in a menu hierarchy, and its expected inputs and outputs.

When a script or plug-in wants to call our plug-in, they pass by the PDB, which transmits the inputs to us, and recuperates the outputs, passing them back to the caller.

A core function that's exposed to the plug-ins will have a wrapper around it in the core, to register it as a PDB procedure, and another wrapper in libgimp to allow the function to be called as if it were a normal function.

So - that's the overview, now down to the nitty-gritty.

3 The Basics

3.1 Behaviour

A GIMP plug-in will behave in one of three ways. It can accept image data, and do stuff to it, returning new image data (this set includes all blurs, edge detection, bump maps, color modification plug-ins). It can accept image data, do something with it, and return nothing (a file save plug-in will typically do this). Or it can take nothing as an argument, and generate a new image. Both file load plug-ins and the various script-fus in the main toolbox behave like this.

3.2 Essentials

MAIN ()

Your plug-in needs a MAIN. In brief, this function registers the callbacks for the init, quit, query and run events. It does this by looking for the...

```
GimpPlugInInfo PLUG_IN_INFO = {
   init,
   quit,
   query,
   run
};
```

This **must** be called `PLUG_IN_INFO`, must have file scope, and must contain 4 function pointers. `init` and `quit` may optionally be `NULL`, but `query` and `run` must have non-null values.

init is called on GIMP start-up, after query. This allows the procedure to do some other initialisation, if required, after registering itself. PSPI uses this function, for example, to query Photoshop plug-ins.

quit is called when the GIMP quits, and a plug-in is still running. This allows the plug-in to clean up after itself, before it gets killed by the system when the parent GIMP process goes down. This is also usually NULL.

3.3 query

query is called at GIMP start-up, and registers the procedure with the PDB. Since this call will typically not change from one invocation to the next, the results of the call are stored in the file pluginrc in .gimp-2.0, and unless the file has changed since the last start-up, the previous values registered for the plug-in will be used.

The general form of the query function is:

```
static void
query (void)
{
 static GimpParamDef args[] =
 {
  {
   GIMP_PDB_INT32,
   "run_mode",
   "Run mode"
  },
  {
   GIMP_PDB_IMAGE,
   "image",
   "Input image"
  },
  {
   GIMP_PDB_DRAWABLE,
   "drawable",
   "Input drawable"
  }
  ...other arguments here...
 };

 gimp_install_procedure (
  "plug_in_my_plug_in",
  "Blurb",
  "Help string",
  "Author",
  "Copyright",
  "Copyright date",
  N_("<Image>/Filters/Misc/"
       "_My plug-in..."),
  "RGB*, GRAY*",
  GIMP_PLUGIN,
  G_N_ELEMENTS (args), 0,
  args, NULL);
}
```

I'll elaborate a little on what the last 7 arguments to `gimp_install_procedure` mean.

N_("¡Image¿/Filters/Misc/_My plug-in...") declares that the procedure will install itself in this menu location with the mnemonic M. Note that mnemonics are not placed on the sub-menus, and that we set the entire path (and not just the end bit) as translatable.

"RGB*, GRAY*" declares the types of images which we accept. The GIMP currently supports RGB, GRAY and INDEXED images, with or without an alpha channel. RGB*, GRAY* means that we accept any of RGB, RGBA, GRAY and GRAYA images.

`GIMP_PLUGIN` declares that this plug-in runs in an external process, and not in the core.

`G_N_ELEMENTS ()` is a handy macro which gets the number of elements in an array of GimpParamDefs. The last 4 arguments of `gimp_install_procedure` declare, in order, the number of input parameters, the numlber of output parameters, the form of the input parameters (an array of GimpParamDefs), and the form of the output parameters.

3.4 run

The other required procedure in `PLUG_IN_INFO` is the run procedure. This is where all the good stuff happens.

The prototype of the run function must be

```
static void
```

```
run (const gchar      *name,
     gint              nparams,
     const GimpParam  *param,
     gint             *nreturn_vals,
     GimpParam       **return_vals)
```

This gives the procedure name being called (required, since a plug-in may register many procedures), the number of input parameters, followed by the parameters themselves, and a pointer to the number of return parameters, followed by a pointer to the parameters themselved (to be filled in by the plug-in).

`return_vals` must always have at least one value returned. The first, and only required) return value must be of type GimpPdbStatusType, and is used by the core to allow error detection on plug-in calls. Usually, this will contain the value `GIMP_PDB_SUCCESS`.

The run function delegates the actual image treatment to a separate function in general, to allow the same function to be used regardless of the way in which the plug-in has been invoked.

3.5 Run modes

A plug-in can be run in a number of dufferent ways, interractively, non-interractively or with the same values as last time.

These run-modes correspond to the plug-in being called from a menu item, being called from a script or another plug-in, or being called using the Ctrl-F shortcut (run last plug-in).

`GIMP_RUN_INTERRACTIVE` is the case where we create an options dialog usually. In the other cases, we will typically call the main routine of the plug-in directly. I won't go into this any further, since there are plenty of good examples in the GIMP plug-ins source code.

4 Accessing Image Data

4.1 GIMP image data structures

A GimpImage is a container class for all of the data in an image, including text layers, vector data, guides and, of course, image data.

This image data is organised in GimpLayers, GimpChannels, GimpSelections and other data structures [1]. Each of these data structures is a GimpDrawable.

A GimpDrawable is, in brief, any GIMP object on which you can draw. This includes layers, selections, masks and channels.

4.2 PixelRegions

The primary means of accessing image data in a plug-in is through GimpPixelRegions. We obtain the bounds of the drawable we're going to mangle (I mean manipulate) and initialise a source and destination pixel region from that using the following:

```
gimp_pixel_rgn_init (GimpPixelRgn *,
    GimpDrawable *,
    x, y,
    width, height,
    dirty, shadow);
```

dirty is a boolean parameter which indicates whether the tiles in this pixel region should be marked dirty for the core, and shadow indicates whether shadow tiles should be used during the operation to improve display speed.

Usually, for read-only pixel regions, we will use FALSE for dirty and shadow, since the tiles will never be dirtied, and we won't need to work on shadow tiles.

Similarly, for read/write operations, we will usually initialise the pixel region with TRUE for dirty and shadow.

4.3 Tiles

The GIMP is a tile-based program. Internally, image data is stored in blocks of 64x64. If you have every had a heavy load on a computer, and been working on a large image in the GIMP, you have probably seen the GIMP updating block by block.

There is an easy way to handle the data one tile at a time, using the function `gimp_pixel_rgns_register`. To use this function, you tell it how many pixel regions you wish to register, and keep in sync during traversal. Then the following code segment allows you to travers pixel regions sequentially.

```
  for (pr = gimp_pixel_rgns_register (2, &s
     pr != NULL; pr = gimp_pixel_rgns_process (pr))
{
  gint row;
  guchar *src = srcPR.data;
  guchar *dest = destPR.data;
  ...
}
```

Note that a pixel region has a number of properties - width and height, bits per pixel, and its data, which is a `guchar *` with `myPR.bpp * myPR.w * myPR.h` bytes of data in it.

Accessing the data for a given pixel is then as simple as accessing the appropriate offset in `myPR.data`.

4.4 RegionIterators and PixelFetchers

Traversing pixel regions as above can be quite repetitive, but it is frustrating, because typically there is always one extra thing or one line of code that you can't do the way it's been done a hundred times before. Thankfully, there are times when you can extract that template repetitive code into a library.

The GimpRgnIterator functions provide a variety of common ways to traverse a PixelRegion, using a pre-defined function pointer per pixel.

There are also a number of neighbourhood-based algorithms which get dramatically slower on region boundaries, to the point where a special treatment for neighbourhoods which are completely inside a tile is called for. The GimpPixelFetcher routines provide a way to hide the special treatment of tile borders, making plug-in code more readable and shorter.

5 Utility libraries available to plug-ins

- libgimpwidgets Helper widgets (usually just collections of widgets with handy callbacks).
- libgimpthumb Thumbnail generation for plug-ins, heavily used by GAP.
- libgimp* Various other libraries used to a lesser extent, including naive colorspace conversion routines, parasite manupulation, etc.

5.1 Compilation and installation

To compile a GIMP plug-in, there is a utility provided by the GIMP called gimptool. With this we can install a plug-in either in the user's home directory (in .gimp-2.0/plug-ins) or in the system plug-in directory. The syntax is:

```
$ gimptool --install plugin.c
```

or

```
$ gimptool --install-admin plugin.c
```

This utility can also be used to uninstall plug-ins, or install scripts.

6 Conclusion

The goal of the talk was to show that the basics of coding a GIMP plug-in are nbot that complicated. You can quite quickly be manipulating RGB values and adding menu entries. This process can be accelerated even more by using a scripting language such as Python or Perl to write your plug-ins. However, this comes at the loss of some functionality because of parts of the libgimp utilities which are still not available to those language bindings.

For those who would like to look for real examples of plug-ins, there are a number of good examples in the GIMP source code itself. Some of the more accessible ones of each type are `edge.c` and `pnm.c` in `plug-ins/common`.

References

[1] The GIMP 2.0 API reference `http://developer.gimp.org/api/2.0/app/index.html`

[2] Kevin Turner *Writing a GIMP plug-in* `http://gimp-plug-ins.sourceforge.net/doc/Writing/html/plug-in.html`

Translation Technology At Sun Microsystems, Inc.

Tim Foster
tim.foster@sun.com
Software Globalization, Sun Microsystems, Inc.

Introduction

This paper aims to show how Sun Microsystems, Inc., uses translation technology for its translation activities on GNOME and other projects.

Primarily, we will share our experiences on the use of open standards such as XML Localization Interchange File Format (XLIFF) and Translation Memory eXchange format (TMX) and the use of tools to process these formats which can increase translator productivity and aid in the sharing of translations across multiple projects.

We will demonstrate our translation editor which has been developed in-house and has been in use for several months on real-world translations. In keeping with our tradition of supporting open standards, our editor can load and save XLIFF files and can export TMX files for use with other translation tools. We have had extensive feedback from professional translators who have been using the system in production and we will explain some of the features that were added to accommodate their needs.

Finally, we will present our vision of translation technology and the advantages it can bring to increase translator productivity and translation accuracy.

Background

In common with most large computer systems companies today, Sun Microsystems, Inc., translates its products into several languages in order to sell to a global market. In this paper, we would like to share some of our experiences in doing large-scale translation projects.

In particular, we've seen that people working on the GTP (GNOME Translation Project) are interested in making improvements to their translation consistency, tools and processes and we believe that our experiences on large-scale translation projects may be helpful. One technology area that we see as vital to increasing productivity for translators and helping to increase translation quality is that of standards. The use of tools that process documents that conform to these standards has already proven to be useful within Sun Microsystems.

The annual translation throughput for Sun Microsystems is 40 million words (that is, we produce 40 million translated words per year) which, by GNOME standards probably doesn't sound like a huge amount, but does take significant amount of resources to produce.

Along with the cost of doing the translations comes the amount of time to produce these translations. A typical translator will process 2000 words per day: which, based on the number of words we translate, amounts to 20,000 days per year. Clearly, with this volume of translations to produce each year we need to have a very efficient process to produce translated products as soon as possible after the English product ships – obviously, as we refine this process, we would like to ship localized products at the same time as the English product.

With the time it takes to do these translations, there's a related problem: that of consistency. As mentioned above, due to the volume of text that needs to be translated, it is obviously too much for one translator to work on (even one translator per language would find this volume an insurmountable hurdle!). Ideally, we would have a team of translators working on a different section of the product at any one time. However, they must make sure to use the same terminology and style of translation as their peers working on other sections of the product; otherwise, the end-user of the product gets an inconsistent user-experience.

So, in order to be able to release our products on time, with good quality and within a limited budget, we had to invest in translation technologies that would allow us to automate and streamline the translation process to maintain or improve translation quality and to improve the efficiency of the workflow.

Translation Standards: Their Need and Availability

In the course of day-to-day translation work, a translator can expect to receive files in any one of a number of formats including but not limited to:

- HTML
- SGML
- XML
- Plain text
- PO files.

The translator must be proficient with a wide variety of editing tools and file formats. If they don't have the software to read DocBook XML, for example, they will have to obtain them, and learn how to use them. Often, gaining the expertise to work with a new file format takes time, which could be better spent actually doing translations. Moreover, people writing tools to process and manipulate the resulting files (for example, to give a calculation of word-count so that they could see how much material has been translated and display those statistics on a web page) requires an understanding of each different file format. In other words, every time a new file format is presented for translation, work is required both by the translator and the project co-ordinator or tools team to ensure that the new file format can fit adequately into the existing project infrastructure – clearly, an undesirable situation. One possible solution to the need to process a wide range of files formats is to restrict files formats to one format or a small number of formats. However, based on our experiences at Sun Microsystems, restricting different product groups to a single file format is an uphill battle.

Jim Waldo, Distinguished Engineer at Sun Microsystems, has some interesting things to say about standards in a recent blog post titled “Why Standards ?”:

> *They [standards] started out solving problems. Because they [standards] solved the problems, people used them. The use drove the standard, not the other way around. This allows innovation, this allows technical progress. Things that work get used by people who are trying to solve problems.*
>
> *http://www.artima.com/weblogs/viewpost.jsp?thread=4840*

The problem we described is an ideal case for a standard. Translators must be able to concentrate on the task of translating software and tools developers need to be able to write a tool once and not have to update it each time a new file format is introduced.

By employing translation standards in our tools, we solve both of these problems. Each time a new format is introduced, a filter must be written to convert that format to XLIFF. After that, all tools that process translations can work with XLIFF documents and do not have to be changed each time another format is introduced. In addition, it means that our translators are free to choose the translation tool that they're most comfortable with – in our case, we have written a translation editor which can read and write files in these translation standards but there's nothing to stop translators using some other translation tool which supports the standard, or even write their own emacs or vim mode if they want to.

The two main standards we are currently employing at Sun Microsystems in translation technology are XLIFF and TMX. XLIFF is a relatively new standard, designed specifically for the task of translation by a consortium of translation vendor companies and computer companies, and now managed under OASIS (Organization for the Advancement of Structured Information Standards). An XLIFF file contains the translatable text from any input document. Once the XLIFF file has been completely translated and reviewed, it can be converted back to the original document format.

XLIFF has support for a number of translation-related processes, such as providing information about the context of the string for translation, support for review-cycles and notes from translator, to name but a few. One of the most useful parts of the XLIFF file format is that it allows for text to be “pre-translated”: that is, suggested translations can be inserted for each text segment being translated. This means that translation memory and machine translation tools can suggest translations for the translator to consider.

The other main standard that we use is TMX (Translation Memory eXchange). TMX was designed to allow translation memories (databases containing source strings and their translations) to be portable across translation memory tool implementations.

There is a subtle relationship between XLIFF and TMX. An XLIFF file contains the state of a translation in-progress whereas a TMX file contains the completed translation which can then be imported into a translation memory.

Our general approach to translation technology

At Sun Microsystems, we believe in the use and promotion of open standards and our translation activities are no exception. Our vision of translation is to employ technologies that improve the efficiency of the translation process – resulting in a faster turnaround for translations and better quality translations. Ultimately, we would like to improve the process to the level of getting “translations out of the wall” - in the same way you access your bank account using an ATM, we would like to be able to translate software by simply plugging the source files into a translation system, and receive the results immediately. We are gradually working towards that goal – today, with the exception of our translation editor, all of our tools are server-side allowing us to deploy the system on very large machines and databases where we can maximize the performance of the system and make the system easier to manage.

Our use of translation technology starts at the point of content creation. Technical writers working on the source language document use a translatability assessment tool. In the same manner that a spell checker would flag incorrect spellings in the source document, the translatability assessment tool will flag sentences that might be difficult to translate according to certain language rules. Each rule contains a sample sentence that suggests a rewrite of the sentence.

The next major piece of technology we use is a portal-based work flow system. This manages the whole localization process: source-file delivery from the engineering teams, followed by conversion to XLIFF and partial translation by other translation tools and the eventual delivery of the partially translated XLIFF files to various translation vendor companies throughout the world. Those vendors connect to the portal, select files and start to work on the translation of the files. Once the translation is completed, the XLIFF files are then uploaded back into the portal which sends the files back to the engineering groups for inclusion in the final product.

We also use a glossary management tool. This stores not only source language terms in a database as well as the translations that have been approved for those terms. Using a glossary management tool allows translators working on different documents to look-up the correct translation for each term, providing a better experience for end-users and a more consistent translation.

We are investigating the use of machine translation tools in our translation work flow. The other major tools in this work flow process are the translation memory software and the translation editor. We'llcover these in more detail in the following sections.

Translation memory system

Our translation memory system is a key component in our translation technology tool set. It consists of two main components. As mentioned earlier, the first component is a large database containing source segments and their translations in a number of languages. Each segment in the database is usually a sentence, but it could also be a paragraph or a phrase. The segments in the database are stored with some meta-data, describing the context of the segment (for example, which source document it came from or which engineering group created it).

The second major component of a translation memory system is a fast fuzzy search mechanism. Using this, we can search through the translation memory for the segments in an input document and produce a partially translated file. Every time we find an exact match in the database, we can store that translation as a suggestion in the output document. We can also look for fuzzy matches in the database. Fuzzy matches which are quite similar to the input segment, but don't match exactly. We can add fuzzy matches to the XLIFF file also, and include data about the quality of the match for the translator, which can be used when they are completing the translation.

When doing such a search against the translation memory, we can also choose to narrow the search using the metadata stored in the database, limiting our search to documents that are in the same category as the input document or have been produced by the same product group. Of course, there's a trade-off between the amount of matches we could achieve versus the usefulness of those matches to the translator – returning several matches per segment may actually slow the translator down as they need to search through those candidate translations for the correct one.

Again, localization standards are used by this tool where possible. The input and output documents for the translation memory are XLIFF documents: the input contains the source language segments, and the output contains the source language segments and suggested translations for those segments in a single target language. Also, the translation memory system can import TMX documents, adding to the number of segments in the database.

Although the system handles XLIFF files, it is useful to be able to keep track of the different formatting styles in the original document. XLIFF allows us to mark formatting sections in the original document and preserve them through the translation process. For a translation memory tool, this is very useful because we can easily take account of formatting when computing how “fuzzy” a match is. For example, the segment:

This is a *<emphasis>*small*</emphasis>* sentence.

is quite similar to the segment :

This is a *<b>*small*</b>* sentence.

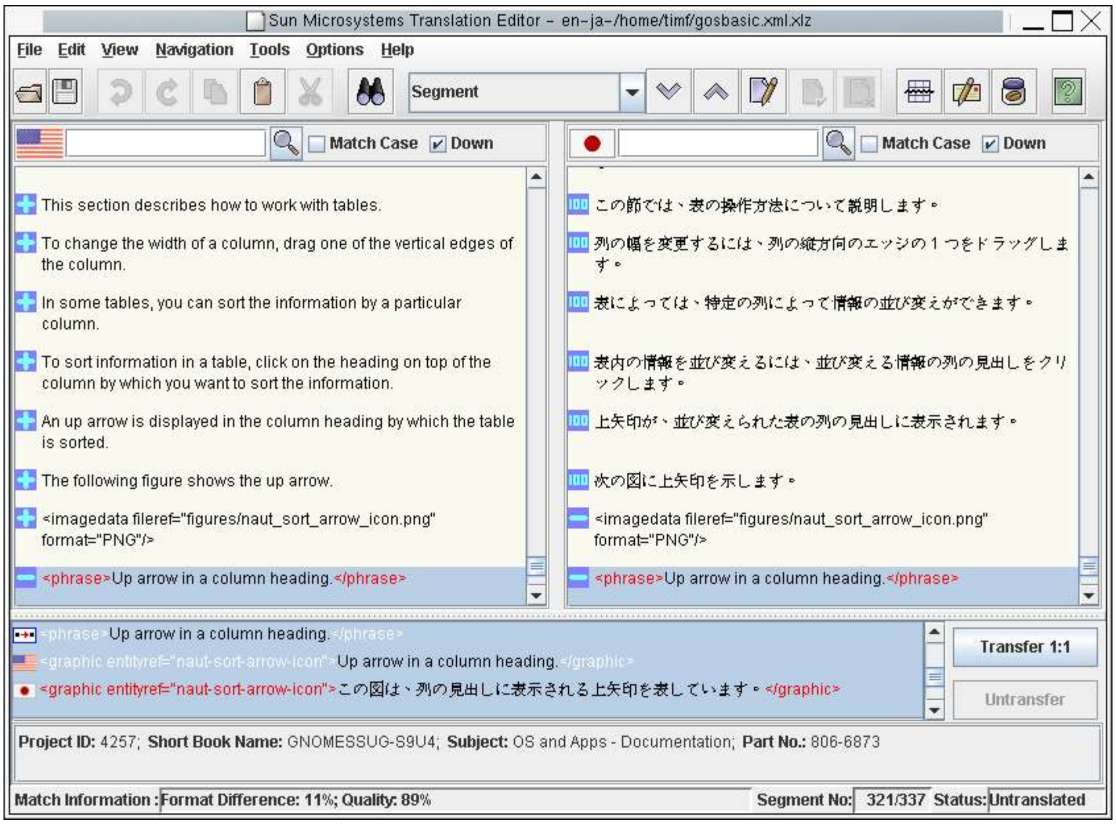
Although the two segments appear to be about 60% similar, if we disregard the formatting in those segments, they are exactly the same. To maximize cross-format leveraging (for example, obtaining matches from a DocBook translation when the source document was HTML) we employ special algorithms to weight matches so that the difference in formatting sections does not influence the final percentage-match figure as much as differences in the content of the strings – these segments would still be considered different, but according to our match algorithm, they turn out to be 96% similar.

As the translation memory system is the first tool we use in our translation work flow, for convenience, we have also built in a set of filters that will convert documents in HTML, DocBook, plain text and XML to XLIFF. We are also working on a number of software-message file filters.

Translation editor

The translation editor is the main interface that translators use to translate XLIFF files.

Illustration 1, an actual screen-shot of the the translation editor, is split into three main sections. Segments in the source language document are shown in the column at the left of the screen. Segments in the target language document are on the right of the screen (although this layout can be changed). The suggested matches for the current segment are shown at the bottom of the screen, along with any meta-data about the match and an indication of the quality of the match that was found. The suggestion area highlights the difference between the match found in the database, and the actual segment for translation. There is a transfer button, which allows the translator to transfer the selected suggestion into the target language area, where they can then complete the translation.


Sun Microsystems Translation Editor - en-ja-/home/timf/gosbasic.xml.xlz
File Edit View Navigation Tools Options Help
Segment
Match Case
Down
Transfer 1:1
Untransfer

This section describes how to work with tables.

To change the width of a column, drag one of the vertical edges of the column.

In some tables, you can sort the information by a particular column.

To sort information in a table, click on the heading on top of the column by which you want to sort the information.

An up arrow is displayed in the column heading by which the table is sorted.

The following figure shows the up arrow.

<imagedata fileref="figures/naut_sort_arrow_icon.png" format="PNG"/>

<phrase>Up arrow in a column heading.</phrase>

この節では、表の操作方法について説明します。

列の幅を変更するには、列の縦方向のエッジの１つをドラッグします。

表によっては、特定の列によって情報の並び変えができます。

表内の情報を並び変えるには、並び変える情報の列の見出しをクリックします。

上矢印が、並び変えられた表の列の見出しに表示されます。

次の図に上矢印を示します。

<imagedata fileref="figures/naut_sort_arrow_icon.png" format="PNG"/>

<phrase>Up arrow in a column heading.</phrase>

<phrase>Up arrow in a column heading.</phrase>
<graphic entityref="naut-sort-arrow-icon">Up arrow in a column heading.</graphic>
<graphic entityref="naut-sort-arrow-icon">この図は、列の見出しに表示される上矢印を表しています。</graphic>

Project ID: 4257; **Short Book Name:** GNOMESSUG-S9U4; **Subject:** OS and Apps - Documentation; **Part No.:** 806-6873

Match Information : **Format Difference: 11%; Quality: 89%** **Segment No:** 321/337 **Status:** Untranslated

Illustration 1 Sun Microsystems Translation Editor

During the course of developing the editor, we sought feedback from our translation vendors and feel confident today that it fits the needs of these users. As with any end-user focused application, the editor had to be extremely simple and intuitive to use. It is easy to configure and provides all of the features that a translator would expect. During consultation with our users, the following features were requested most frequently, and have been implemented in the current version of the editor:

- Format checking – formatting that is in the target segment can be checked against the formatting in the source segment, to ensure the translator hasn't omitted or included any extra formatting (bold text, italic, etc.).
- Integrated "mini TM" - when the translator encounters a segment that has been previously translated in the file, it will be translated automatically. Also, any sentence that has been translated in the past that is similar to the one currently being translated will also be suggested as a match.
- Translation status marker – translations can be marked as "complete", "for review" or "approved" to aid in the translation/review process.
- Translation comment marker – a translator can add comments about any segment which may be useful in the translation/review process.
- TM match information – each translation memory suggestion contains information about how close the match is to the source segment and also displays any meta-data that was stored with the proposed translation.
- Printing – we can produce a HTML formatted version of the XLIFF file at any time for printing and review.
- Easy navigation – commonly used keyboard shortcuts can be modified to the suit the translators preference.
- Back-conversion – for ease of use, the translator can choose to convert the XLIFF file back to the original file format so they can check the layout/formatting of the finished translation.

Future of translation technology at Sun Microsystems

Despite the translation tools that we already have in production at Sun Microsystems, we feel that there is still work that could be done to further improve the translation process. With XLIFF, we have an excellent way to add more translation tools into the translation process. We would like to integrate the tools we have at the moment so that the translator can use a single interface to all of the translation tools.

As mentioned earlier, XLIFF allows us to provide translation suggestions from more than one tool. We envisage a workflow where a translation memory tool would add the first set of translation suggestions to the XLIFF file. A next stage in the process would be to perform “term-mining” on each segment. This would identify possible terms for translation which could be automatically looked up in the translated glossary, and these terms could be highlighted by the translation editor. A final stage of the translation workflow would be to pass each segment into a machine translation system and add those translations into the XLIFF file. All of this work would be completed before the file reaches a translator.

On the client side, we believe that a networked group of translators could be more efficient than a group of standalone translators. We would like to investigate additional features for the translation editor that would create a peer-to-peer network of translators, so that translations completed by each translator are made available to any other translator connected to the network, thus improving the efficiency of the group.

The standards that we are using at the moment make this work easier, but other standards will be important as we add new tools into the translation process.

TBX is a standard file format for transfer of glossary information between translation tools. OLIF (The Open Lexicon Interchange Format) is a standard for transfer of terminology between machine translation and other NLP tools. We believe that these standards will also play a part in our vision of a translation workflow.

A final effort worth mentioning is the one being worked on by the OASIS Translation Web Services TC which will further enable tools from different groups to work together.

Conclusion

In this paper, we have shared our experiences with translation standards and have expressed our belief that the use of common standards in the translation space is worth the investment. We hope that the many people working on free and open source software will find them of benefit to the different projects that they are working on. We have also demonstrated the use of a dedicated, standards-based translation memory system and shown some of the features in an standards-based translation editor that are useful to translators when translating software and documentation.

Resources

XLIFF : http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=xliff

TMX : http://www.lisa.org/tmx

OLIF : http://www.olif.net

TBX : http://www.lisa.org/tbx

OASIS WS TC : http://www.oasis-open.org/committees/trans-ws/charter.php

Sun Global Application Developer Corner Article on XLIFF:
http://developers.sun.com/dev/gadc/technicalpublications/articles/xliff.html

A Math-Canvas for the Gnome Desktop

Luca Padovani[*]
Department of Computer Science, University of Bologna
Mura Anteo Zamboni, 7 – 40127 Bologna, Italy

Abstract

In this paper we describe GtkMathView's features for the creation of interactive, customizable applications involving rendering of mathematical markup. We also give an overview of the GtkMathView's internal architecture and suggest some of the most promising future developments of the widget.

1 Introduction

GtkMathView[1] is a GTK+ widget for rendering of and interaction with MathML markup [10]. Although free interactive applications for math are available, none of them can be easily treated as a light-weight, customizable component. GtkMathView aims at filling this gap by providing a reusable piece of software that can be embedded wherever an application needs to display mathematical formulas and possibly provide interactive functionalities to the user.

The need for a specialized widget for rendering mathematics should be evident: mathematical notation is half-way between plain text and free-style graphics. Formatted mathematics usually makes use of glyphs provided in fonts for which no standard encodings exist, hence using already developed components for the layout of text, such as Pango [1], is of little or no help. Moreover, as we have shown in [3, 4], math formatting is particularly context-sensitive and a large number of parameters must be taken into account: formatting quality, font availability, physical characteristics of the output device are only a few of them. The design of an architecture for math formatting is thus a challenging and interesting problem for which we have found an elegant solution that GtkMathView implements.

[*]Partly supported by the European Project IST-2001-33562 MoWGLI.

[1]See `http://helm.cs.unibo.it/mml-widget/`

The strength of GtkMathView as a component is potentially its most severe weakness: mathematics rarely occurs "stand-alone", it is usually embedded within a larger document. Although the integration of GtkMathView with other widgets (for text or HTML layout) might seem like a straightforward completion, this is more easily said than done. Lots of technical issues arise as soon as one is interested in mixing different markup languages, still preserving natural and flexible exploitation of the displayed information. For this reason GtkMathView understands a simple extension of MathML which allows the embedding of mathematical expressions within simple text layouts. The implementation of the support for this extension has had the welcome side-effect of highlighting parts of GtkMathView's architecture which are not tied to MathML formatting, thus improving the overall modularity and organization of the widget's source code.

This paper is aimed at giving an overview of GtkMathView, in terms of both its Application Programming Interface for those developers who want to use it in their own applications (Section 2), and also of its internal architecture for those developers who want to extend it, possibly porting it to different platforms, enabling it to use different font families, embedding it within specific applications (Section 3). In Section 4 we list some of the most important developments scheduled for the near future, and we conclude in Section 5 with some final considerations.

2 Standard API

2.1 Markup for Mathematics

GtkMathView renders MathML documents. MathML is a standard XML application for the encoding and the representation of mathematical expressions. MathML comprise a content-oriented markup as well as a presentation-

```
<math>
  <mfrac>
    <mn>1</mn>
    <msqrt>
      <mn>1</mn>
      <mo>+</mo>
      <msup>
        <mi>x</mi>
        <mn>2</mn>
      </msup>
    </msqrt>
  </mfrac>
</math>
```

$$\frac{1}{\sqrt{1+x^2}}$$

Figure 1: A simple MathML document along with a possible rendering.

oriented markup. The latter is the only MathML markup understood by GtkMathView. Figure 1 shows an example of MathML presentation markup, along with a possible rendering

MathML is meant to encode mathematical expressions only and while these may occasionally be communicated or processed in isolation, it is more often the case that they appear in the context of a larger document. In fact, MathML can be naturally embedded within foreign XML markup, (X)HTML being one of obvious candidates. In addition to MathML, GtkMathView understands an XML markup language called BoxML which can be used for the rendering of simple documents with embedded mathematical formulas. BoxML provides for very simple formatting primitives (boxes) as well as "holes" where MathML markup can be plugged in. Table 1 lists the elements of BoxML along with a short description of their rendering semantics. BoxML elements must be used in their reserved XML namespace (`http://helm.cs.unibo.it/2003/BoxML`).

2.2 Models and Views

GtkMathView provides a *view* for a mathematical document encoded in MathML (possibly intermixed with BoxML) markup. Such a document must be encoded somehow in order for GtkMathView to create an internal data structure that is suitable for formatting and eventually rendering. Until version 0.6.2 the only way of feeding GtkMathView with a document was to use GMetaDOM [8, ?], a compliant implementation of the Document Object Model [7] built on top of Daniel Veillard's libxml2 library. Recent development versions of GtkMathView offer a greater flexibility in the choice of the source document model: it is now possible to use libxml2 tree directly, or libxml2 reader interface, or the user may provide an adaptor for its own document model. The only requirement is that it must be possible to make a pre-order traversal of the model data structure.

As GtkMathView provides a view of the supplied model, operations on the view should normally be reflected on the model. For this reason each model can provide a notion of *model element identifier* which is used by the widget for optimizing the process of incremental formatting and for the communication between GtkMathView and the host application through GtkMathView's API. In most cases such identifiers are just pointers to the nodes of the model data structure.

2.3 GTK+ Interface

The GTK+ interface provides a set of *high-level* methods for the creation, the usage, and the destruction of instances of the GtkMathView widget. A summary of the main methods and signals provided by the interface is given in GtkMathView's reference manual. GtkMathView's interactivity support can be summarized in three aspects: *selection*, *point-and-click*, and *editing*, which we examine next.

Selection By *selection* we mean the possibility for the user to distinguish one or more model elements from the others in a MathML document. The typical visual feedback for selected elements is that of displaying them with a different background color.

Selection support in GtkMathView consists of three methods and four signals. The methods are needed to set, reset, and query about the selection status of a particular MathML element, as this information is hidden within the internal data structures of GtkMathView and is not available as part of the MathML document itself. Selection is a boolean property: an element can be either selected or not, there are no multiple levels of selection.

Methods for selection and de-selection operate recursively on the model structure. The de-selection method can be used to perforate a previously selected parts thus leaving "holes" within a selection. This mechanism can be

Table 1: Summary of BoxML elements.

Element	Attributes	Description
`box`		root element for BoxML markup
`action`	`actiontype`, `selection`	represents a number of alternative renderings. `actiontype` is the type of action; currently only `toggle` is supported. `selection` is the 1-based index of the child to be rendered by default
`at`	`x`, `y`	renders an element at the specified coordinates. This element is valid within a `layout` element only
`h`		renders its children aligned on their baseline.
`ink`	`color`, `width`, `height`, `depth`	renders a solid box of specified color and size
`layout`	`width`, `height`, `depth`	renders a fixed layout box; each child must be an `at` element
`space`	`width`, `height`, `depth`	renders an empty box of specified size
`v`	`enter`, `exit`	renders its children vertically. The first child is the topmost one, the last child is the bottommost one. The entry and exit baseline of the whole element can be controlled by the two corresponding attributes. Their default value is 1 and −1 respectively, meaning that the entry baseline is taken to be the one of the first child, and that the exit baseline is taken to be the one of the last child
`text`	`color`, `background`, `size`	renders the text contained by the element with the specified color and background
`obj`		renders a fragment of MathML embedded in BoxML

exploited for representing *patterns* of elements within the model documents.

The GTK+ signals related to selection are called `select_begin`, `select_end`, `select_over`, and `select_abort`. The first three signals provide, among other information, the model identifier of the MathML element on which the signal has been emitted. The signals are fired in disjoint sequences matching the following regular expression:

```
select_begin
    (select_over)*
    (select_end | select_abort)
```

which is to be read as follows: selection begins when the user presses the first mouse button and moves it a bit from the original position (`select_begin`). As the user moves the mouse with the button pressed, an arbitrary number of `select_over` signals is emitted. Selection can terminate in two cases: either the user releases the button (`select_end`) or she presses another mouse button, hence aborting the selection (`select_abort`).

An interface for which multiple selections are required can check the status of control keys on the keyboard in order to determine whether a new selection sequence replaces a previous selection or adds a new selection to it.

Support for semantic selection The widget does not highlight automatically the parts of the model on which the user is dragging the mouse. It is complete responsibility of the application to handle the selection signals and to invoke the selection method accordingly. Although this puts some burden on the application side, it also enables the maximum flexibility, as selection may be constrained in ways that, in the most general case, are infeasible to hard code within GtkMathView.

Point-and-click Functionalities Point-and-click is supported by the `click` signal, which is emitted when the users clicks on the view. Again the signal carries the identifier of the model element on which the user has clicked.

Among the possible usages for this signal are the activation of hyperlinks and the management of `maction` elements. When the user clicks on the view, the signal handler can look for a model element that has a `xlink:href` attribute set, and open that document. The search is typically done by starting from the element provided by the signal, and possibly climbing up the chain of elements until one with the required attribute is found, or the root element is reached, or any other application-specific condition applies.

Activation of `maction` elements works in a similar way. Currently only the `toggle` action is supported by GtkMathView. In particular, once the `maction` element is found, it is possible to increase the number in the `selection` attribute that determines which of the `maction`'s children is displayed. If `selection` is increased beyond the actual number of children, GtkMathView recasts it into a valid range via a modulo operation. This way the application code for handling `maction` does not have to know the exact number of children, and is consequently simplified. Since this behavior is not mandated by the MathML recommendation, it can differ in other MathML rendering engines. Once the `selection` attribute is set with the new value, GtkMathView will recompute automatically the document's layout.

Possible conflicts in case the `xlink:href` is set on an `maction` element must be resolved by the signal handler.

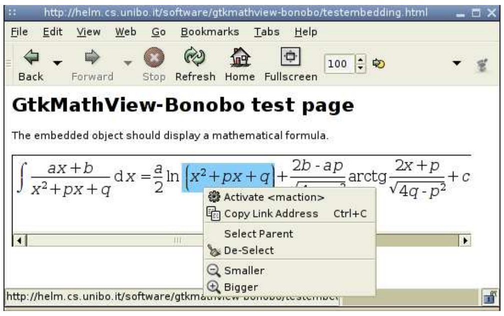
Editing The notion of *editing* in the context of GtkMathView is very limited. It simply refers to the fact that GtkMathView tries to react incrementally when the source document model changes. GtkMathView does not enforce any constraints on how and when the model can change.

GtkMathView implements a number of internal mechanisms that try to optimize rendering, in the sense of minimizing the amount of computation that is needed to re-render a document after a modification has occurred.

In some cases local modifications of the model may have non-local effects. For instance, modifying the content of a table cell may cause the re-computation of the whole table layout, as MathML attributes for table can specify constraints among cells in different columns or rows.

2.4 Bonobo API

There is a Bonobo Component [2] for GtkMathView which allows developers to embed instances of GtkMathView within Bonobo applications. The component implements the


http://helm.cs.unibo.it/software/gtkmathview-bonobo/testembedding.html
File Edit View Web Go Bookmarks Tabs Help
Back Forward Stop Refresh Home Fullscreen
100
Activate <maction>
Copy Link Address Ctrl+C
Select Parent
De-Select
Smaller
Bigger
http://helm.cs.unibo.it/software/gtkmathview-bonobo/testembe

GtkMathView-Bonobo test page

The embedded object should display a mathematical formula.

Figure 2: GtkMathView embedded within Galeon showing a partially selected formula.

`PersistFile` and `PersistStorage` interfaces, as well a GtkMathView-specific interface that resembles very closely the GTK+ API discussed in the previous section. The main issue of the Bonobo component is the management of the document model. When using the GTK+, the document model is a data structure shared between GtkMathView and the host application. GtkMathView reads this data structure and communicates identifiers on this data structure. However, when GtkMathView is used as a Bonobo component, there is no guarantee that the component and the container run in the same address space. Consequently it is not possible in general to share data structures, and model identifiers implemented as physical pointers are not feasible any more. One solution would be to provide Bonobo interfaces for the document model (in fact, DOM already provides these interfaces), but this would result in a significant overhead. For now the Bonobo component just embeds the document model, which is thus unaccessible by the host application except for the communication of numerical identifiers. Although this solution is largely unsatisfactory, no other reasonable ways has been found.

One interesting application is to use the GtkMathView Bonobo component together with the `mozilla-bonobo` plugin[2], so that GtkMathView can be used as an external plugin for rendering mathematical formulas (see Figure 2). Even though Mozilla and all the other Gecko-based browsers have native MathML support, this happens to be extremely slow especially when lots of tables are involved, so GtkMathView may provide a more efficient alternative.

[2]See `http://www.nongnu.org/moz-bonobo/`

3 GtkMathView's Internals

GtkMathView operates like a compiler. It parses a source language, the MathML document to be rendered, it creates an internal tree representing the parsed document, and it "compiles" this tree into a lower language (called *area language*) that is suitable for display. Accordingly to this structure the GtkMathView's source code has been organized in the following way:

`src/common` basic types and definitions (colors, characters, strings, smart and weak pointers, fixed-point numbers);

`src/frontend` parsing code;

`src/frontend/common` parsing code independent of the source model. All the parsing logic for both the source document and the configuration files is confined in this directory;

`src/frontend/custom_reader` frontend for parsing a document using C functions.

`src/frontend/libxml2_reader` frontend for parsing the source document using the libxml2 reader interface.

`src/frontend/libxml2` frontend for parsing the source document when it is represented as a libxml2 tree.

`src/frontend/gmetadom` frontend for parsing the source document when it is represented as a GMetaDOM model.

`src/engine` the source-independent and device-independent formatting engine;

`src/engine/common` markup-independent code (formatting tree base classes, view class);

`src/engine/mathml` MathML-specific formatting classes;

`src/engine/boxml` BoxML-specific formatting classes;

`src/engine/adapters` classes for mutual embedding of MathML and BoxML markup;

`src/formatter` the backend;

`src/formatter/common` definition of the formatting context;

`src/formatter/areamodel` device-independent area model;

`src/formatter/mathml_graphic_device` device-independent formatting rules for mathematics;

`src/formatter/boxml_graphic_device` device-independent formatting rules for boxes;

`src/formatter/gtk_areamodel` GTK+ implementation of the area model;

`src/formatter/gtk_mathml_graphic_device` GTK+ implementation of the formatting rules for mathematics;

`src/formatter/gtk_boxml_graphic_device` GTK+ implementation of the formatting rules for boxes;

`src/widget` implementation of GtkMathView's widget API.

3.1 Frontend

Like said, the frontend is responsible for creating the internal formatting tree corresponding to a source MathML document. The problem is, the way this document can be accessed may broadly vary. For a long time we have advocated the use of GMetaDOM but this solution has some problems: it introduces a number of dependencies (in fact two more libraries) with respect to the plain libxml2 library; it also introduces a computational as well as a memory occupation overhead, since its structures are wrappers of libxml2 objects; it is not as successful as we hoped it to be, thus making its shortcomings unbearable to most GNOME developers. Moreover, the nature of GtkMathView as a component should not prevent the possibility for the source MathML document to be encoded in some application-specific models not necessarily implemented using libxml2.

It is thus important for GtkMathView to be able to adapt its frontend. This is in contrast with the fact that the frontend must be very efficient, since MathML documents tend be grow rapidly even for simple expressions and an efficient frontend is a crucial key for incremental rendering. The solution currently adopted makes large use of C++ templates for factoring the *logic* of the frontend, that is the sequence of operations that builds up the internal formatting tree and that are independent of the source document format. The implementation of a new frontend entails the definition of some classes among which the most important are:

- a `Model`, defining the basic data types of the source document model including strings and nodes, as well as conversion functions from model strings to GtkMathView strings and iterators over nodes of the source model;

- a `Linker`, which is used for keeping the source model and the formatting tree linked in both directions. This is not required (the linker may act as a no-op class) but it is important for interactivity and incremental rendering.

As we have anticipated in Section 2, recent development versions of GtkMathView provide a *custom* frontend that is completely independent of any document model. All the user has to provide is a set of callback functions that GtkMathView calls when performing a pre-order visit of the document model.

3.2 Backend

GtkMathView's architecture has been designed in such a way that it is possible to easily port it to different platforms. By "platform" we mean the combination consisting of the Operating System, of the available fonts, of the set of characteristics expected from the formatted document (that is, whether it is meant to be printed on the paper or on the screen, whether it is meant to be editable or static, and so on). In this sense it is more appropriate to say that GtkMathView's can be *adapted*, not merely ported, to different contexts. The point is that the formatting of mathematical notation is a much more delicate and complex task than formatting plain text because the context affects formatting in numerous and subtle ways. Adaptability is not achieved by simply instantiating areas with platform-specific drawing primitives, so the actions taken by the formatted must be reworked consequently.

Adapting GtkMathView to a different context entails the development of:

- a set of *platform specific areas* for drawing glyphs and possibly colored rectangles on the drawing window;

- one or more *shapers* for mapping Unicode characters into suitable combinations of glyph areas and possible other area types. As mathematics often requires the use of fonts with non-standard character encodings and the composition of glyphs for representing stretched characters (parentheses, brackets, arrows, integrals, summations, products, wide accents, are just a few examples), shapers already provided by the platform are of little or no use.

- a *graphic device* that knows how to format the basic mathematical constructs. This cannot be platform independent since many parameters that control formatting depend on the used fonts, on the expected quality of the formatted document, on the resolution of the output device.

Currently only the GTK+ platform is supported, meaning that specializations of the aforementioned classes specific to GTK+ are provided.

4 Future Developments

Among the most relevant aspects for which we are going to provide active support we mention:

- Implementation of an OCaml [6] frontend that allows GtkMathView to read a document model encoded as an OCaml data structure. In the past we have been using GMetaDOM as the interface between C++ and OCaml, but as we have explained this solution is not completely satisfactory. By allowing GtkMathView's frontend to access directly the OCaml data structure, we hope to achieve a more efficient, light-weight integration.

- Implementation of a backend for rendering MathML documents to PostScript. This feature has been available in the past up to version 0.5.3 of GtkMathView, but it has not been restored yet in the 0.6.x series, which implements GtkMathView's adaptable architecture.

- Implementation of a backend for Win32 and, more generally, finalize a porting of GtkMathView for the Windows platform. Various attempts have been done in the past, but none of them resulted in a completely satisfactory product. Now that issues related to the frontend and the backend have been neatly separated in GtkMathView's architecture, it is much easier to accomplish this task.

- Integration of GtkMathView within other GTK+ widgets and other applications. Although it is possible to use GtkMathView as a Bonobo component, or to embed GtkMathView within a GTK+ text widget, the embedding always compromises most of the flexibility allowed by the "naked" GtkMathView widget. Hence it would be desirable to use GtkMathView as a library for rendering mathematics within larger applications. Among them we cite GtkHTML, KHTML, and AbiWord.

- Development of an equation editor based on GtkMathView.[3]

5 Concluding Remarks

Since GNOME is now growing to a fairly mature desktop environment, proper support for rendering and editing mathematics is now becoming an important feature and GtkMathView seems to be a promising candidate for becoming a standard component that fills this niche.

With a careful design and implementation of GtkMathView's architecture we have tried to accommodate the needs of the largely varied GNOME community, especially in those cases (the frontend of applications dealing with XML markup) where no standardized solutions have clearly emerged. At this point we would be glad to see more enthusiasts joining us in an active effort for providing GNOME with a state-of-the-art, competitive and open-source math-canvas based on GtkMathView.

References

[1] Owen Taylor, "Internationalization in GTK+", 1999, `http://developer.gnome.org/doc/whitepapers/gtki18n/`

[2] Michael Meeks, "Introduction to Bonobo: Bonobo & ORBit", `http://www-106.ibm.com/developerworks/webservices/library/co-bnbo1.html`, August 2001

[3] L. Padovani, "MathML Formatting", Ph.D. Thesis, Technical Report UBLCS-2003-03, Dept. Computer Science, Bologna, Italy, 2003.

[4] L. Padovani, "MathML Formatting with TEX Rules and TEX Fonts", to appear in TUGboat, The Communications of the TeX Users Group, Volume 24.

[5] L. Padovani, "Interactive Editing of MathML Markup Using TEX Syntax", to appear in the Proceedings of the International Conference on TEX, XML, and Digital Typography, 2004.

[6] Xavier Leroy and Damien Doligez and Jacques Garrigue and Didier Rémy and Jérôme Vouillon, "The Objective Caml system release 3.07 Documentation and user's manual", `http://caml.inria.fr/ocaml/htmlman/`

[7] Arnaud Le Hors and Philippe Le Hégaret and Gavin Nicol and Jonathan Robie and Mike Champion et al. (Eds), "Document Object Model (DOM) Level 2 Core Specification", Version 1.0, W3C Recommendation, November 2000, `http://www.w3.org/TR/DOM-Level-2-Core/`

[8] Raph Levien, "Design considerations for a Gnome DOM", informal note, 28 March 1999, `http://www.levien.com/gnome/dom-design.html`

[9] Paolo Casarini, Luca Padovani, "The Gnome DOM Engine", in Markup Languages: Theory & Practice, Vol. 3, Issue 2, pp. 173–190, ISSN 1099-6621, MIT Press, April 2002.

[10] Ron Ausbrooks and Stephen Buswell and Stéphane Dalmas and Stan Devitt and Angel Diaz et al.: Mathematical Markup Language (MathML) Version 2.0 (2nd Edition) W3C Recommendation, (2003), `http://www.w3.org/TR/2003/REC-MathML2-20031021/`

[11] Donald E. Knuth: The TEXbook, Addison-Wesley, Reading, MA, USA (1994)

[12] Leslie Lamport: A Document Preparation System LATEX, Addison-Wesley, Reading, MA, USA (1986)

[3]A prototype editor based on TEX/LATEX syntax for input of mathematical notation called EdiTEX has been developed, but it is by no means complete. See `http://helm.cs.unibo.it/editex/` for more details.

GNOME AND THE ENTERPRISE USER

Ghee S. Teo <ghee.teo@sun.com>, Brian Nitz <brian.nitz@sun.com>

Sun Microsystems Inc.

Abstract

Sun announced the release of its Sun Java Desktop System (JDS)[tm] in late 2003 with a plan to deliver between 500,000 and 1 million copies of this desktop to China. This paper presents a behind the scenes view of the various problems faced and solutions adopted in the creation of an enterprise desktop based fundamentally on Open Source software. The primary targeted audience is project managers, engineering managers, CTO, and others who plan to distribute and contribute to Open Source Software.

1 Introduction

The first section of this paper will cover the development of a GNOME desktop for enterprise users. It explores the development process as well as decisions and compromises made during this process. While it may be interesting to put together a desktop like the Sun Java Desktop System (JDS), the litmus test for the success of such an endeavor is what happens when it is deployed to real users. Therefore, in the second section we share our initial experiences in the deployment of this desktop. We will discuss lessons learned from desktop pilots within Sun and to external customers. It presents some of the problems faced by end users and discusses possible solutions.

2 Decisions Decisions

The first step of the development was to choose the constituents for the complete desktop.

2.1 Choosing the base Desktop

The choice of a base desktop was actually made in August 2000 when Sun chose GNOME as a replacement for the vintage CDE[tm] desktop on Solaris[tm]. As Sun's desktop team worked to deliver GNOME on Solaris, they gained valuable experience and grew a relationship with the community. Customers and employees also became familiar with GNOME, so this seemed the obvious choice on which to base a new enterprise desktop (JDS). The engineering team was given the go-ahead to productise the GNOME desktop and they were given a tight deadline. Sun saw two major strategy factors in a fast time to market. One was the maturing of GNU-Linux desktops, the other was Microsoft's software assurance policy which drove many enterprises and even governments to seek alternatives. There is a trade-off between developing close to community head and developing against a stable base. Solaris GNOME development introduced us to the downside of development against HEAD, desktop behaviour can change significantly from build to build. As there was little time to adjust for this in our short release cycle, the JDS product team based the first release on a stable GNOME build. At the time this decision was made (April 2003) GNOME 2.2 was the latest stable release. Sun's Accessibility team continued to work close to HEAD so that their fixes would go into the community and future GNOME desktops on all platforms.

2.2 Choosing the base Operating System

The Linux Java Desktop System is part of a project which Sun plans to deliver on Linux, Solaris X86 and Sparc platforms. It made sense for Sun to partner with an existing GNU/Linux vendor to achieve a quick time to market in the GNU/Linux. Given that SuSE[tm] was the second largest commercial Linux vendor and its location in Germany suited the Sun's development team well.

2.3 What other applications and libraries should be included?

Some of us would like to include as much as possible on our own desktops. We might have several chat clients a few dozen games, three or four browsers, a couple of editors and so on. But in order to manage support and training costs, enterprise customers have quite a different wish list. Due to supportability, enterprise desktops can often be quite minimalist. They might be limited to a browser, an email client, a calendar and an office suite. Sometimes, they might condense everything to a browser and a logout button.

2.3.1 Choosing the browser

The choice of browser can be a contentious issue. Many companies are constrained by market requirements and preferences of existing customers. Technical discussion on the validity of the browsers almost becomes a moot point in the final decision. Netscape[tm] was Sun's default browser until AOL[tm] decided to drop Netscape development. Mozilla then became the natural choice for the Java Desktop System. Mozilla 1.4 is a stable and feature rich browser compared with the version of Netscape that Sun previously delivered. Therefore much of the effort in the browser delivery was in assuring that the release was well integrated with the desktop and that various plugins were also included.

2.3.2 Choosing the Mail and Calendar Client

The decision of which email client to include was simple, Evolution provides a user interface familiar to most Windows users. One of the objectives of the Java Desktop System is simplifying migration and training. Using Evolution as an Email and Calendar client can significantly reduce the cost of re-training of users.

2.3.3 Choosing the Office Suite

A decent, complete office suite is a killer application for the enterprise desktop. Sun's open sourcing of StarOffice[tm] facilitated the creation of the OpenOffice.org open source community. Sun's Star Office team works closely with the OpenOffice.Org community to create a feature rich office suite that runs on many platforms including Linux, Solaris and Microsoft Windows[tm]. Sun's release of StarOffice 7[tm] is bundled with and integrated into JDS. StarOffice 7 and OpenOffice.org 1.x both import many legacy formats and save files in a compatible and open XML format. This will help provide an escape route from proprietary format and OS lock-in that is so prevalent today.

2.3.4 Other trade-offs and compromises

This brings us to another issue all open source vendors face. The use of patent tainted and licensed codecs can prevent GNU/Linux from making inroads on the desktop. Open source vendors are faced with three choices:

- Integrate software which may violate patents or licenses
- Integrate nothing and let the user figure out how to integrate various codecs
- Integrate software from vendors who are willing to license to GNU/Linux

Sometimes we must balance long-term goals against the need to provide a better user experience in the world as it is. Hence, third party offerings including Adobe Acrobat Reader[tm], RealPlayer[tm], Macromedia Flash(tm) player are integrated into JDS, but many codecs from those who are not amiable to GNU/Linux are not included. Legally questionable codecs were removed from gstreamer. To achieve a more cohesive Java run-time and browser plugins, JRE 1.4.2 is an integral part of the first release of Sun's Java Desktop System.

3 Problems, Solutions, Problems, Solution

3.1 Building the GNOME desktop on SuSE

Personally the most exciting part of the project was to see how an Open Source project such as GNOME is built from scratch. Creating the Build Environment evolved over a number of stages, each targeting a specific problems. The stages are:

- Development of Build Environment – resolve the basic tools for building the desktop
- Distribution of Build Environment – allow other development teams to build on JDS
- Automation of Build Environment – discover early any build problems

3.2 Development of the Build Environment

This is the first phase of the development, we need to be able to build the GNOME desktop on a SuSE distribution. We needed to determine whether the development tools available on the selected SuSE distribution were capable of building the targeted version of GNOME. We followed these steps:

1. Install the selected version of SuSE distribution which was SLEC 8.1[tm]
2. Install all the development tools required
3. Download the latest GNOME 2.2 tarballs
4. Create spec files to build the GNOME desktop components
5. Build the desktop components from bottom up using modules dependency as specified in jhbuild
6. Install and note any additional build tools and dependencies required
7. Install the built components on the build machine
8. Repeated steps 4 to 6 until the complete stack is built

When the cycle is completed we have the complete set of build tools and versions required to build the basic desktop. Apart from documentation build tools, every component built smoothly on our selected version of SLEC. Several cycles helped us to resolve subtle run-time errors, such as those caused by the quirkiness of libtool.

3.3 Distribution of the Build Environment

One of the key lessons Sun learned from GNOME 2.0 development on Solaris is that the build environment can be difficult to replicate. Even slight differences in versions of build tools can cause a build failure. Since our developers are distributed around the world, this could hamper the development efforts. Fortunately, there are two important elements in the base OS which simplify the management of the build environment. First of all, the rpm database provides a record of the name and version of the rpm installed and used. Initially, the engineering team had to write scripts to automatically pull in all the developer's rpms. Once the build environment was stabilised, the build tools were integrated into the development tool category in YaST2 and developers could install the complete build environment by the click of a button.

3.4 Automation of Build Environment

Once the basic desktop is up and running, test engineers begin searching for bugs and developers begin to pull in various patches and features from GNOME HEAD, as well as developing Sun's value added features. Release engineering wrote scripts to automate the build and generate build reports. This is a important part of the build environment as it provides a quick feedback to any build patches almost hourly instead of waiting for the nightly build report. Since build report is sent to the developers, problem could be rectified immediately. The nightly build is also used by the developers to speed up bug fixing activity. Since developers provide spec files to build the rpms, a complete record of the build steps are also recorded. These allows other developers to build any other modules without much insight into the specific components.

4 Branding the desktop

Our primary targeted customers were enterprise users who are familiar with Microsoft Windows. JDS branding was designed to provide similar experience to these users, and yet different enough that people wouldn't be shocked or confused when they did encounter differences. When you move from a Ford to a BMW you expect differences, yet the brakes and the steering wheel should be in the same place!

4.1 Branding the Panel

- The default setup provides only a single panel at the bottom
- The Launch menu provides access to almost anything one needs (exception is nautilus)
- The name of the distribution is shown on the side of the launch menu pane

4.2 Blueprint theme

Sun developed a default theme called 'Blueprint' which depicts the JDS unique look and feel. Blueprint is an image based theme. A new theme engine was developed to allow the colorization of images with a given set of colors. The colour tones for a given image can be easily altered with the Blueprint GTK engine. The colourisation also applicable to the stock icons. To give a much smoother look for the desktop, Blueprint theme also provides rounded combo boxes and text buttons.

4.3 Administration tools integration

Yast2 provides system administration commands. Instead of having the user navigate through the Yast2 user interface, many of these commands are now available through either System-Settings or System Tools. The gnome-vfs vfolders framework together with the “Categories” in the desktop files are used to wrap the yast2 commands. Yast2 was also branded with Sun's colours.

4.3 Extra Bits

We also integrated extra bits to improve the user experience, these include:

- CUPS printing as developed in XD2
- nautilus printing as resulted from an internal usability study
- world clock which arose from the needs of internal users
- JRE
- Java Applications from Java Open source community
- Browser Plugins
- Extra Localisation
- VNC based secure remote desktop (only in Release 2 and later)
- Configuration Management (only in Release 2 and later)

5 The Litmus Test

Now we come to the true test of a desktop, what do users think? End users and administrators expect certain features of any desktop. These are Stability and Performance, Usability, Integration and Interoperability. Enterprise users have additional expectations, these include security and scalability. This section explores lessons we learned from internal pilots of GNOME on Solaris and the Java Desktop System.

5.1 Stability and Performance

JDS did quite well here, especially considering that JDS 1.0 was being compared to existing desktop operating systems that are well past version 1.0. Desktop glitches did occur, but system lockups and other bugs that prevent users from doing their job were quite rare. Modern hardware makes it difficult to judge performance, but I haven't seen many reports indicating that this is a problem. Our UI team helped us decide when to add throbbers to reassure users when no obvious action was taking place.

5.2 Usability

An Irish hospital was one of the first pilots of the Java Desktop and when their end user encountered JDS, the first question was “Why doesn't it say 'START?'” Clearly we had met our goal of not frightening end users with unfamiliarity. It's interesting that most of the usability complaints I encountered were from system administrators.

- ***Consistency:*** The YaST tool used OK or Apply, while gnome configuration settings were all instant apply.
- ***Task Focus:*** Some menus and YaST configuration options were organised based on engineering or marketing abstractions rather than user tasks.
- ***Inappropriate access to desktop properties***: One user found a simple and all too obvious

way of rendering all nautilus icons invisible. He associated the mime type of icons with his application.

- ***Personal preferences:*** A few people didn't like the theme. It is difficult to find a theme that everyone can agree on. Fortunately this is easily configurable in GNOME.

5.3 Integration and Interoperability

This presents one of the biggest challenges to Open Source expansion onto corporate desktops. Administrators and end users have expectations about interoperability. Often they don't even know that they've fallen into the trap of relying on proprietary data formats or network protocols.

5.3.1 Network Interoperability

Nautilus goes quite a long way toward providing seamless Windows network integration. Customer reported issues with some Windows shares led us to discover possibilities for improvements in future versions of JDS and other GNOME -based desktops.

5.3.2 Application Interoperability

JDS 1.0 did not integrate WINE, so out of the box it does not run .EXE files. Some might consider this a deficiency, others might consider it a feature. We hope users will consider replacing Windows only applications with GTK or Java based applications.

5.3.3 Data type Interoperability

Integrated Realplayer, Flash, Adobe Acrobat and other plugins helped provide this capability. The StarOffice/OpenOffice.org XML file format makes it possible for documents be shared between users on many different operating environments. For example, this document passed between a user on JDS and another on a Sunray[tm] and it is readable by a user running OpenOffice.org on an Apple's OSX[tm]. Universally readable document formats were common in the past, there is no reason why we shouldn't have them in the 21st century. However we did discover a couple of glitches:

- Some configurations of Microsoft mail clients and servers will wrap otherwise readable document types such as RTF, ASCII or JPEGs, into Application/MS-TNEF streams. This is much like uuencoded binary except that integrated MS-TNEF extractors do not exist on many popular operating systems including Apple OSX, GNU/Linux. Even Windows 95 users might encounter problems decrypting attachments from users of these products. Unfortunately, the sender is led to believe they are sending a universally readable data format.
- Webservers will occasionally send StarOffice/OpenOffice.org files with mime hints of application/zip. It is up to the client to figure out that the file is actually the compressed XML format of StarOffice or OpenOffice.org document.

5.4 Security and Scalability

There were few surprises regarding security. System administrators were impressed with the desktop lockdown capability and JDS is not vulnerable thousands of the worms and viruses which now pollute the internet.

Our experience with GNOME on Solaris and SunRay taught us a few lessons about scalability. Two examples come to mind. The first was Nautilus's insistence on searching for trash folders on mounted filesystems. This was O.K. when the user had one disk and a few dozen directories under root on a single local hard disk. But when Nautilus encountered deep NFS filesystems shared over worldwide networks, it caused some performance issues.

The second scalability example has to do with scalability on thin clients such as SunRay. One of our engineers was testing a new Solaris utility called dtrace when he noticed that something was destroying pixmaps quite frequently. He traced it to a GNOME panel applet with a scrolling text display and found that this applet was madly creating and destroying hundreds of pixmaps per second. Xservers generally don't like unnecessarily destroying pixmaps, but when the applet was running on a single user's machine with the graphics buffer and CPU sharing a local bus, it didn't cause too much damage. But when hundreds of users ran the applet on a single box with their displays connected to the CPU through a 100MB ethernet... the performance hit was noticeable. We now have fast computers and fast networks, but we shouldn't completely ignore the performance issue when it has the potential to impact scalability.

6 Conclusion

JDS was targeted towards a specific type of enterprise worker, however it has gained acceptance in areas beyond this initial goal. There is room for improvement, but GNOME has grown into a mature desktop that is a viable option on the corporate desktop. I would like to thank other members of the open source community for helping to make this possible.

WvSync

Tea, Earl Grey, Hot: Replication with WvSync

Dave Coombs <dcoombs@nit.ca> *Net Integration Technologies, Inc.* 2004 05 16

1 Disclaimer for Flamers

This paper contains many exaggerations, numerous factual errors, and several outright lies, because I estimate I have written less than half of the code I will be demonstrating next month at Guadec. For now I hope to distract you with whimsical rhyming headings.

Readers are encouraged to check `http://open.nit.ca/wiki/?page=WvSync` for updates, corrections, complete rewrites of this paper, etc.

2 Expository Story

One day, a day like any other, Avery and I were MBWAing our R&D office in Montreal. We noticed something that hadn't really seemed clear before: something ridiculous like 75% of our developers were working on some form or another of data replication. People were writing backup code; other people were backing up these backups elsewhere; a group of people were co-operatively writing code to synchronize calendar and contact information between Outlook, Evolution, and Horde; other people were synchronizing user accounts and DNS records between servers; someone was replicating MySQL databases; someone else was synchronizing files, and one truly crazy individual was working on a distributed filesystem.

These are all largely separate projects, but these people were all doing largely the same thing, with different bugs and/or design flaws and/or tradeoffs in different places.

We decided this was crazy.

This paper describes some universal requirements for any replication project, the inherent compromises involved, and puts forth an "optimal compromise" leading to a general solution that should be useful in any project.

WvSync is the start of an open-source implementation of this solution to the general replication problem. I will describe some design elements of WvSync, hopefully without lying too much.

3 Replication Elaboration

3.1 Things You Need, Guaranteed

In any project involving replication, the requirements are bound to be pretty similar. So far as we can tell, they'll always look something like this:

- I have some data in some format all in one place.
- I have big performance requirements, and/or I need reliability, and/or I want to conserve bandwidth, and/or I want to reduce latency.
- Therefore I should have multiple copies of the data, or different parts of the data, in different places.
- Therefore I somehow need to choose which data to replicate, how many times, how often, and to which places.
- And now I need some software to actually do the replication in my particular format.

Individually these requirements may not seem that bad, but put them all together and it starts to become rather implementation-specific, dependent on what you are trying to do with what particular kind/size/shape of data.

So everyone may have these requirements, but everybody does something different with them in their own project, as follows.

3.2 Reproachful Approaches

The compromises begin. Compromises are fine, of course, but keeping it generalized is hard. Given the above requirements, the approach taken for any given project will vary, but is always the result of constraints on a combination of only the following:

- CPU power
- Storage space / memory
- Network bandwidth
- Network latency
- Reliability requirements

For example, a backup tool has constraints on CPU power and storage space, and it may choose to waste storage space to conserve CPU power. A network backup tool is restricted mainly by network bandwidth. A calendar/contact synchronizer is restricted by client-side storage *and* network bandwidth/latency, and might choose to waste space to reduce latency. A distributed filesystem is severely restricted by network bandwidth and latency. A web cache should probably waste network bandwidth (and storage, of course) and even perfect reliability to reduce latency. These are all compromises that trade away something (usually disk space or bandwidth, which are cheap) to achieve something more important to the user.

Eventually the compromise usually comes down to deciding which data you keep a copy of ("make available offline" in Windows-speak) and which data you'll go to the server for when you need it ("don't make available offline"). Some protocols implement only one or the other. Simple caching is in between, but mostly doesn't continue to work when you go "offline."

3.3 Optimize the Compromise

We believe there is, in fact, a general solution to the replication problem with an optimal set of tradeoffs, in which you don't have to choose betwen "online" and "offline".

"But there can't be an optimal tradeoff!" you say. "Otherwise, why would you use different tradeoffs with each project?"

Because we're lazy, and we didn't want to implement the whole general algorithm *each* time. Implementing the most practical tradeoff for a particular project is less work, if you're only working on one project. Now that we have a whole slew of replication projects, it makes sense to implement the right solution once. Like all optimal tradeoff solutions, it involves cheating.

The optimal solution needs to follow these rules:

- Replicate a "set of named objects" of variable size. URLs are named objects. So are files. So are MySQL records. So is everything.
- Provide a way to convert the named objects into the things we actually want, like URLs, files, or actual MySQL records.
- Provide a special API for accessing the replicated objects whenever anybody wants them. (This is the hardest part, but we can skip it initially, so that's OK.)
- Waste all idle bandwidth. Differentiate between *critical* operations (which the user or application is actively waiting for) and *non-critical* operations (which nobody asked for, but we can do if we're bored). This allows us to use negative latency.
- Waste all available storage space. Cache *everything* and let the replication protocol tell you when your cache is invalid. Find a way to flush your cache when somebody important needs your storage space. Feel free to remember all sorts of metadata to help with decisions later.
- Waste all available CPU time. Take note of which data is used most often, and use this to decide what to flush from the cache and when, and what to replicate, how many times, and how soon. Use something like librsync to waste cheap CPU cycles instead of sending whole objects whenever a tiny change is made.

4 Solution Execution

WvSync is a general purpose replication library written in C++, using WvStreams (for easy network communication etc), UniConf (for easy metadata storage), librsync (for easy rsync-like deltas), and DaZuko (because it's neat).

Picture rsync, and add the following:

- Bidirectional synchronization. Change files on one computer, change other files on another computer, and it will copy the changes in both directions.
- Synchronization of arbitrary data types, not just files. You have to write some C++ glue to support your own data type, but it's well abstracted, and not that hard.
- Online mode. When WvSync is up and running, any changes made will be *instantly* copied to the other computer(s).

4.1 Embrace the Interface

The lowest-level class in WvSync is the `WvSyncObj`, which is a basic interface for a named generic object that can be replicated. You can derive one for a file, a database record, a hash-table dictionary, or whatever, but you'll have to provide your own methods for serializing and deserializing into and out of a `WvBuffer`.

`WvSyncObj` itself provides wrapper functions that use librsync to produce signatures, deltas, and patched reconstructions of the data.

As of this writing, `WvSyncFile` is the only existing useful subclass of this. It overrides/implements the following methods:

- `getdata()` reads a given number of bytes from a given offset of the object, and returns them in a `WvBuffer`. This is used when calculating the librsync signature of the object, and when reconstructing the object if a delta patch is received.
- `findlastmodtime()` finds and returns the last-modification timestamp of the object. This is easy in the case of a file; it's the mtime. This is used for making decisions about what needs to be sent in what direction.
- `findmeta()` finds and returns metadata for the object. In the case of a file, I'm using the mode, the uid, and the gid, formatted into a string. For your own data type, it can be whatever you need. `applymeta()` is related; it takes metadata received from a remote node and applies it to the local object.
- `makecopy()` makes a copy of the current object under a new name. This is used in case of a synchronization conflict. As described below, one node will have its object renamed, and making a copy lets us take advantage of librsync deltas.

4.2 Precision Decisions

In general, we need to send an object from one WvSync node to another if it has changed on the first node since the last synchronization. How do we know if it has? We have to keep a list *for each synchronization partner* of every single object, and the status of that object as of the last synchronization with that partner. Useful things to remember for each object are the last-modified time, the md5sum of the librsync signature, possibly even the librsync signature itself (it's large, but it allows for negative latency in some cases), and the user-defined metadata for the object. Clearly this uses a lot of space and CPU time, but it provides usefulness and minimizes bandwidth usage and latency.

It is the `WvSyncArbiter` class's job to determine what needs to happen. For any synchronizable object, the `arblocal()` method first determines what has happened to the object since the last synchronization. Possibilities are:

– `ok`: Nothing has happened since the last synchronization.

– `new`: The object has been added.

– `mod`: The object has changed.

– `del`: The object is no longer there.

– `meta`: The object contents are the same, but the metadata has changed.

For each object, the applicable one of these five states is sent to the other node, along with current librsync signature, metadata, and last-modification time. The other node does the same calculation based on the object *it* has by the same name, and sends back its version of the same data.

Then the `arbremote()` method is called on both sides. The goal of this function is to arrive at the same decision on both sides.

Here are the cases to consider, assuming WvSync nodes named A and B:

– A `ok`, B `ok`: Nothing to synchronize. Go on to the next object.
– A `ok`, B `new/mod`: B wins; copy to A.
– A `ok`, B `del`: B wins; delete on A.
– A `ok`, B `meta`: B wins; apply metadata to A.
– A `new/mod`, B `new/mod`: Classic conflict case. See below.
– A `new/mod`, B `del`: A wins; forget the deletion and copy to B.
– A `new/mod`, B `meta`: A wins; forget new metadata and copy to B.
– A `del`, B `del`: Nothing to do. Remove from both lists and continue.
– A `del`, B `meta`: Better to be safe, so copy back to A with new meta.
– A `meta`, B `meta`: Conflict case. See below.

There are numerous special cases too confusing to cover here. (What if a file turns into a directory? How should a failed transfer affect an object's stored state?) There is magic to deal with these, but it's more important to understand the basics.

4.3 A Fix for Conflicts

A conflict arises when the same object has been changed differently on both ends of the synchronization.

The goals for conflict resolution in WvSync are twofold:

1. Don't lose anybody's changes.
2. Make it obvious if there was a conflict.

If we don't do number 1, then we are rudely clobbering people's changes and annoying them. If we don't do number 2, then people will get confused and clobber each other's changes by accident, annoying each other. The only workable solution has to accomplish both of the above goals.

The short answer is: when two people change the same object, pick a winner, and rename the other one to a unique name. *Both* objects, the one with the real name, and the renamed one, will show up on *both* ends being synchronized. The renamed object name could (depending on the type of the object) include the server name and timestamp that it was changed, so the two users who made the changes can co-operate and merge the changes themselves. This is the only reliable way.

Which version is the "winner"? It's almost an arbitrary decision. We choose the one with the later timestamp.

4.4 Protocol? None at all!

The protocol used by WvSync hasn't exactly been designed yet. Sorry.

5 Future Features

- Step 1: Finish implementing the above for file synchronization, and make a standalone program that does just that.
- Step 2: Add a sort-of "online" mode after a full synchronization, where any changes to files (use DaZuko!) or objects are noticed and synchronized immediately.
- Step 84657: Provide the special API for accessing objects in both online/offline modes. Cache like crazy. Pull a librsync-based distributed filesystem out of my ear.
- Step 3275284654: I'm rich!

6 Download the Code

You can get WvSync from its project home page at:

```
http://open.nit.ca/wiki/?page=WvSync
```

7 Acknowledgements

Man, nothing rhymes with acknowledgements.

Thanks to Avery Pennarun for defining the General Replication Problem and Solution.

Thanks to Martin Pool for librsync, and Andrew Tridgell for doing all the math.

Thanks to Pierre Phaneuf for harassing me about renames.

Digital Photography in the GNOME environment

Hubert Figuière <hfiguiere@teaser.fr>

May 14th 2004

1. Introduction

Digital Photography in the last years has been the most dynamic market for photography products. Ever since the first consumer products that appeared in the mid 90s, UN*X users have been attempting to use them with their favorite environment, mostly by performing reverse engineering.

A few open source projects have given photographers ways to access their photos off either digital cameras or scanners from UN*X.

In this paper I'll try to provide background and find the current problems when it comes to GNOME integration. In my talk, I'll try to explain possible solutions to those problem.

2. Acquiring

Inputing photos to the computer is the first step towards working on photos with computers. There are 2 ways to perform this operation: using a digital camera and using a scanner.

Digital Cameras

Digital cameras are probably the most popular tools in use for photos input. A variety of manufacturers offers a variety of models, and they all provide support for Windows, most for Mac OS, and to date, none provide support for UN*X systems, not even with binary drivers. Since the first models appeared on the market, hackers succeeded in reverse engineering to make them work with their favourite operating system.

Communication protocol

To establish communication between the camera device and the computer, several communication protocols have been developed by the manufacturers, most of them being proprietary and not publicly documented.

Cameras that connect to serial ports were all different, and for each manufacturer, one had to figure out what was the protocol used to download pictures off the camera storage [3]. Sometime, a few manufacturers bought chips or software to the same OEM vendor, that allowed a few drivers to be more versatile that the others. That was the case for Nikon, Olympus and Epson products that were all using OEM parts from Sierra Imaging. Sometime, we could get the documentation from the manufacturer like Kodak.

Then went USB. USB provided a faster communication pipes to the computer, and USB 1.1 provided a few standardized protocol. The first USB connected cameras were offering most of the time dual USB/Serial connectivity, and the protocol used over USB was usually the same as over serial, making it easy to adapt existing drivers to work with these newer models.

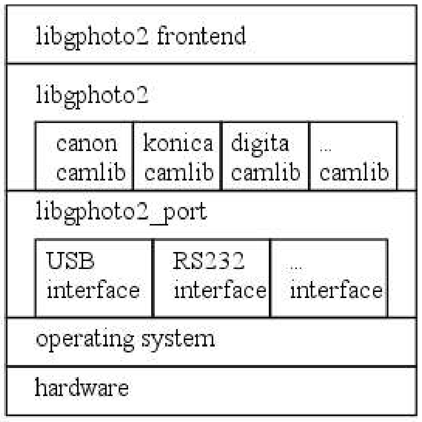
After manufacturers finished transitioning to complete USB connectivity, and after their products matured, standard protocols from USB began to be used more widely: *mass-storage* and later *still image devices* (also known as PTP).

While USB Mass Storage compatible devices are supported natively by Linux 2.4 and other system supporting USB 1.1, simply by mount a filesystem, all the other cameras required some specific drivers, including for PTP, the standard USB Still Image device class.

gphoto

Created in 1999, gphoto has become the de-facto universal swiss army knife tool to access photo from digital camera. The idea was to provide one program for every camera instead of requiring different program for each model available like it was.

When it became gphoto2 in 2000, its internal architecture radically changed to become even more versatile. Everything is now architectured around a library that is modular with different libraries for different cameras, and APIs to write front-ends [1][2].



$Id: architecture.fig,v 1.1 2002/08/17 23:09:55 hun Exp $

camlibs are the libraries that control the camera. This is what you should write or modify to support a new model. They use the *libgphoto2_port* library which abstracts all the I/O through a layer. Current we only have USB and serial support, but there shouldn't be any problem to provide IEEE1394 (aka Firewire) or BlueTooth support later.

libgphoto2 is itself the gphoto2 API. It calls the proper library for the specified device and abstract file retrieval as well as image capture and options setting.

Writing a new front-end is basically calling the APIs from libgphoto2. We currently provide *gphoto2*, a command line front-end, and *gtkam*, a GTK+2 graphical application.

libgphoto2 is currently not meant to provide access to cameras that can simply be mounted like an USB hard disk (USB Mass Storage devices), which represent a majority of the device sold between 2001 and 2003, and still in 2004 [3].

Scanners

Scanners have been around since mid 80s. They are currently much cheaper than digital cameras, but does not seems that popular for photography use. They are mostly designed to scan flat documents like sheets of paper or printed photos, some, with transparency adapter can scan negatives strips or slides. This is good for scanning archives, but if you want to work quickly from shooting to digital publishing, it is not the best option.

Currently there are very few solutions for scanning under Linux. Almost no manufacturer provide specifications or drivers for Linux, but Epson.

Communication protocols

Unlike for digital cameras, there is no standard protocol either de-facto or not, for scanners. This leads to a bigger development effort. Thanks to the OEM game, only chipsets must be supported as most manufacturers seems to buy them to make their own box. It is not easy to guess what chipset is in a scanner without disassembling it.

Parallel scanners are the harder to support due to lack of traffic analyzer on the parallel port, unlike it is for SCSI, USB and Firewire. Some are just SCSI over parallel ports, some have an USB version that do parallel over USB, allowing to snoop the protocol for reverse engineering.

SANE

SANE stands for *Scanner Access Now Easy.* The project has been around since December 1996 to provide support for scanner devices on UNIX. It offers drivers for a lot of different scanners through different libraries.*libsane* provides the APIs [4] to write front-ends, like *xsane* and *xscanimage.*

Unlike TWAIN, a API really popular in the Windows and Mac OS world, SANE makes the difference between user interface and scanner device drivers. This allow SANE to implement scanning over the network with the *saned* daemon.

Image Scan!

Image Scan! is quite an atypical example. Image Scan! is Espon Kowa own scanner driver for Linux. It is available as Open Source for almost everything but a few bits, like some image processing and drivers for some OEM scanners that Epson sells. Therefore it is limited to Linux on Intel platform. The interesting part is that Image Scan! almost implements SANE APIs making the open source driver usable with plain SANE. And Image Scan! could be modified to use SANE APIs correctly so that the front-end can be used as a SANE front-end.

There is goodwillingness from Epson to continue to develop this program, but there don't seem to be any political wills to push their OEM manufacturers to Open Source the rest of the drivers. Still better than nothing, and the world would be better if all manufacturer where as open as Epson is.

Vuescan

Vuescan is the proprietary and shareware solution to scanning on Linux/x86 (as well as on Windows and Mac OS). It offers support for a lot of scanners models, even some that SANE do not have. It can be your last chance solution. Vuescan also provide good support for scanning negatives as its post-processing routines are much better than what SANE front-ends provide.

3. Managing Pictures

After acquiring all your photos, you probably need to manage them. There are several tools. Amongst them, for GNOME, you have gThumb and F-Spot.

If you want to edit them, there is the famous Gimp. No need to talk about this wonderful photo retouching software.

4. Integration in GNOME

We have SANE and gphoto2 presenting the building blocks for universal scanner and digital camera support, but how do they integrate with GNOME ? Beside Gtkam and Xsane front-ends being written using Gtk+ toolkit, and their availability as Gimp plug-ins, there is not much more.

The first point is getting these to use *libhal* [7]. libhal is not by itself GNOME specific because it target cross-desktop interoperability being part of the *freedesktop.org* initiative. It is one of the key parts of GNOME plug-and-play device support, part of the *Project Utopia* [5], providing hardware abstraction layer.

Digital cameras

Robert M Love's *gnome-volume-manager* [6] not only provide high-level support for removable storage devices, like playing CD and DVD discs automatically, but it also provide a means to automatically perform and action, like copying photos, when a digital camera is plugged in. Currently it is limited to cameras that behaves like a disk drives, i.e. those that are recognized as *USB Mass Storage* devices [3] and whose content is available as a mounted filesystem. There is some effort to be done on libgphoto2 side to provide support for these mountable camera and use gphoto2 as the "digital photo import utility". That way, we would have automatic photo import when plugging a supported camera ; or any other action the user might prefer.

The other point would be writing a Gnome-VFS plug-in that provide a filesystem like view of the files in the camera using libgphoto2. The problem is not that simple, as to provide concurrent access to the VFS, we need to serialize all the camera operations by using a daemon that gets exclusive use of the device. This solution is much more elegant even in term of usability, but much more complex to implement.

Scanning

GNOME does not provide an API to allow acquiring a single image. This is an area that should definitely be worked on.

5. Conclusion

There is still a lot of work to go to have a complete integration for digital camera and scanner high-level support in GNOME. Sure the building bricks are still evolving, but the user side it is not as friendly as one might wish.

6. Bibliography

1. Christophe Barbé — *gphoto2 Foundation for a GNOME photo management application* — http://ufies.org/~christophe/gphoto2/ presented at *Boston Gnome Summit,* July 19, 2002

2. Tim Waugh, Hans Ulrich Niedermann, Michael J. Rensing — *The gPhoto2 Manual* — http://www.gphoto.org/doc/manual/, November 2003.

3. Hubert Figuière — *Digital Camera Support for UNIX, Linux and BSD* — http://www.teaser.fr/~hfiguiere/linux/digicam.html, May 2004

4. The SANE project — *SANE Standard Version 1.03* — http://www.sane-project.org/html/, February 22nd 2003

5. Robert M Love — *Project Utopia* — http://primates.ximian.com/~rml/blog/archives/000395.html, blogged on April 7th 2004

6. Robert M Love — *The Linux Kernel and The Linux Desktop* —

http://tech9.net/rml/talks/rml_fosdem_2004.sxi, presented at FOSDEM 2004 in Brussels, February 21st 2004

7. David Zeuthen — *HAL Specification 0.2* — http://freedesktop.org/~david/hal-0.2/spec/hal-spec.html, December 22nd 2003.

Integrating OpenOffice.org in GNOME

Stephan Schäfer
Sun Microsystems GmbH
s.schaefer@sun.com

Dan Williams
Red Hat, Inc.
dcbw@redhat.com

Oliver Braun
Sun Microsystems GmbH
oliver.braun@sun.com

1 Abstract

OpenOffice.org is a full fledged, international, open-source office productivity suite. It is available on a large variety of platforms, including Linux, Windows, Mac OS X, FreeBSD, and many more. Only tiny portions of the code base are platform dependent and subject to actual porting effort.

Though portability has its merits, it does come with a price: lack of integration into the user's desktop environment. This becomes most obvious in the visual appearance of the application. Despite using the system's colors and user interface font, the basic appearance is the same on all platforms, never really sharing the look and feel of other applications designed for the desktop.

We present a solution to bring the GTK+ look to OpenOffice.org without compromising portability. By directly attaching to the system's theming engine, our approach preserves OpenOffice.org features like accessibility and Right-To-Left user interface as required for Arabic and Hebrew localized versions. As a result, OpenOffice.org will follow the current desktop theme on the targeted platforms, including effects like mouse-over, pre-light, and control focus, as well as support for on-the-fly theme switching.

Further integration improvements will allow to import parts of the system configuration. This includes the printing environment, mailer, and browser settings, as well as settings for fonts and proxies.

Additionally, OpenOffice.org extensions to Nautilus will make meta data like author and version information available in the document's properties dialog.

2 The Native Widget Framework

2.1 Introduction

Like any other software package providing multi-platform support, OpenOffice.org uses a concept of system abstraction. With regards to the user interface, only the most essential interfaces to the operating system require porting effort. This includes the creation of top level system windows and the drawing of pixels, lines, bitmaps, and text into them. To receive any kind of input either from the user or from the operating system support for receiving notifications of events must be provided too. The individual user interface elements like widgets or menus are implemented in a platform independent way on top of this system abstraction layer. The described functionality is provided by the visual class library (VCL), one of the base modules of OpenOffice.org[1].

The result of this rather minimalist approach is a uniform look and behavior on all supported platforms and a limited porting effort. However, having a uniform look contradicts with most users' demand for seamless integration into their desktop environment. Two possible solutions would be to either extend the interface of the plat-

1 http://gsl.openoffice.org

form layer to include native widgets, or to emulate the look and feel of the target platform. For example, various Java language APIs provide both approaches, with AWT drawing only native widgets, and Swing drawing everything by itself and only emulating the desktop theme. An example of a hybrid approach, combining the lightweight controls found in Swing with native widgets, is given with the Standard Widget Toolkit[2] (SWT) provided by IBM.

2.2 Native Look & Standard Feel in OpenOffice.org

The main goals of the Native Widget Framework were to provide as much native look and feel as possible, while changing as little of the platform-independent layers of OpenOffice.org as possible to preserve API and binary compatibility. Wrapping actual instantiated native widgets, while a supported future extension, was not possible given the goals of the current project. Furthermore, native widgets' look and feel differs between platforms, even down to the button order in dialogs. Accommodating this behavior would require much greater changes to the OpenOffice.org code and was not possible in a moderate time frame.

Therefore, the NWF utilizes the native system's theming engine to draw an image of the desired widget in a given bounding rectangle with a given state. Thus, the same rendering code is used as a native widget would do but without the need to deal with a native instance of it. While this is not the optimal approach to obtain a completely native look and feel, theming is the best alternative at this time. Future effort may be directed towards even more “native” integration.

The interface to the platform-native layer was extended slightly to provide methods that call the platform's theme engine to render controls, perform hit-testing, query for support of individual widgets, and retrieve hints about native widgets' composition. The original, platform-neutral widget drawing code is still active as a fallback whenever the platform does not support drawing a certain widget drawing request. In all other cases, the paint request is now forwarded to the platform's widget drawing method, including information such as the bounding box of the original VCL widget, and state information like focus, mouse-over, pressed or disabled. Hit-testing and other user input is still handled by non-native methods, which has not been a problem so far. In fact, the paradigm of user interaction with widgets on the targeted platforms (Gnome, KDE, Windows XP, Mac OS X) is still consistent.

2.3 Implementation

The widget drawing method is part of the same interface that is used for any other graphical output, and therefore the parameters will undergo all transformations normally applied to graphics primitives. The most important transformation is transparent support for UI mirroring required for Arabic and Hebrew locales. Coordinates for drawing as well as those received from the system (mouse and window positions, paint events) are mirrored to always reflect a standard left-to-right user interface to the application code. This applies to native widget rendering as well.

Another important issue is clipping support on system window level. The native widget drawing code must obey the clipping region previously set by VCL to make sure images of native widgets will not overwrite explicitly excluded regions. Unfortunately, the GTK+ Style API for drawing native widgets (e.g., `gtk_paint_flat_box()` etc.) only supports a rectangular clipping region that is passed as a parameter. There is no way to use one's own graphics context already set with an arbitrary clipping region. Instead, the widget rendering calls use their own graphics contexts that are not exposed by any API. The current clipping region is used directly only if it is a single

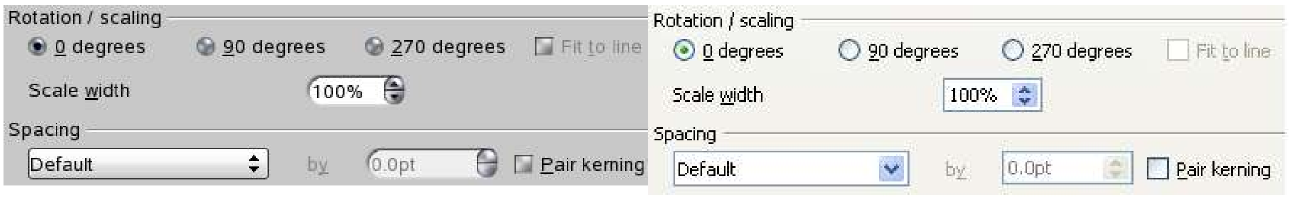
2 http://www-106.ibm.com/developerworks/library/j-nativegui/

rectangle. For complex clipping regions, a more elaborate operation must be performed, consisting of rendering the whole widget (without clipping) into a bitmap and then blitting it to the screen using VCL's internal graphics context and proper clipping.

Modern desktop themes often use visual styles with rounded corners or even work with translucency effects. To correctly deal with partially transparent widgets it is mandatory to fully restore the background before any widget drawing operation may occur. Otherwise, transparent painting would pick up undefined background colors and the same widget might be painted in different shades. Because there is no API to ask a theme if it is partially transparent, rendering of large widgets may be an expensive operation. For example tab pages typically fill a complete dialog and represent the background for many other widgets. Mouse-overs and changes in the prelight state always trigger an expensive redraw of the entire tab page, especially expensive for themes with transparency or complex textures or gradients. By introducing a pixmap cache that is filled once with the uncovered image of the tab page the drawing performance could be increased dramatically. The cache stores the size and state of the widget together with its image, so it is typically initialized upon opening a dialog. Subsequent drawing operations in the dialog then will use the pixmap directly instead of performing expensive redraw operations.

The major difficulty in the GTK+ implementation of the NWF was the variation between themes. Since each theme draws the same widgets slightly differently, using different combinations of state, flags, widget size, and parent widget type, much effort was put into emulating the behavior of the GTK+ toolkit before calling the theme code itself. While this was successful for most themes, some like Crux are obviously broken and require changes of their own.

To correctly deal with different approaches in widget design among multiple platforms certain features can be queried before issuing the native widget drawing calls. Examples are different layouts of drop down list boxes or spin fields. Some platforms require the buttons to be placed inside the edit field but others want them aside. The layout code may be subject to future changes if new target platforms require a widget composition that cannot be achieved with the current implementation.



Differences in widget design between GNOME and Windows XP

For easier distribution, release engineering and maintenance only a single binary of OpenOffice.org should be required for a given target platform. Directly linking against GTK+ libraries or any other toolkit would require an individual installation set for each of them. Despite binary compatibility, the executables could not be launched if the required system libraries were not available. In order to stay with the single binary approach a plug-in concept was introduced by moving the implementation of VCL's system layer in an own component for every toolkit. During startup the plug-in that corresponds to the current desktop environment is loaded on demand and provides the whole process direct access to the underlying platform toolkit. This in fact makes the same binary a native application for multiple toolkits.

3 Native Installer

OpenOffice.org 2.0 will be released using platform's native packaging format. Therefor the whole packaging mechanism has been changed to generate e.g. rpm spec files while staying compatible with the old installer as long as possible. However, the goal is to replace the old installer entirely, which makes repackaging a lot easier or even completely unnecessary.

The most important prerequisite for using native installers was the removal of the separate installation step for each user. Files and directories that are required on a per user base are now created or copied on the fly in the users home directory, which makes OpenOffice.org 2.0 a much better citizen on Unix / GNOME then any previous release.

4 Reusing System Settings

4.1 UI Language

In multi-lingual installations, the OpenOffice.org user interface language will change appropriately when the user chooses a different locale at login, using English as a fallback for languages where the corresponding resource files could not be found. It will be even possible to have different UI and document languages, which is quite popular in Asian countries.

4.2 Personal Preferences

When activating an external hyperlink in a document, OpenOffice.org 2.0 will pass the URL to GNOME and thus the same internet browser application will launch as it would by opening this URL in nautilus.

The OpenOffice.org configuration system has been extended to allow desktop-wide user settings to be accessed transparently from within the Office code. An implementation for GConf allows OpenOffice.org to read proxy settings, personal data like name and address and potentially more settings directly from the corresponding GNOME preferences. Due to a lack of a messaging API in GNOME, even the preferred e-mail client application will be extracted this way.

4.3 fontconfig and CUPS

OpenOffice.org 2.0 dynamically detects if fontconfig is present on the system and if so it transparently merges the fonts supplied by fontconfig with its own fonts.

Very similar to this, OpenOffice.org 2.0 will automatically detect all printers and printer descriptions supplied by CUPS on systems which have CUPS installed.

5 GNOME-VFS support

An optional gnome-vfs based module has been built which interfaces to the OpenOffice.org abstraction of accessing documents (the Universal Content Broker, or UCB). This allows OpenOffice.org 2.0 to open files directly from SMB shares and other gnome-vfs supported URI schemes.

6 Nautilus Extensions

OpenOffice.org 2.0 will save thumbnails of the first document page in the OpenOffice.org 2.0 document package, which allows to write a lightweight nautilus extension to extract and render those thumbnails where appropriate.

Another extension makes meta information of OpenOffice.org documents like author, title, subject etc. available from the properties dialog.

It is also planned to contribute to a new GNOME search and indexing system to ensure the user can search within OpenOffice.org files as well.

7 Acknowledgments

Thanks to Jan Holesovsky for supporting the native widget implementation and its port to KDE, to Philipp Lohmann for designing and implementing the plug-in concept, to Michael Meeks and Andreas Bille for the GNOME-VFS support, and to Sarah Smith for GConf support in the OpenOffice.org configuration.

Graphic Design for Programmers

Liam Quin, XML Activity Lead, W3C

Abstract

Programmers often don’t get exposed to graphic design theory; it’s easy to imagine it’s all about wishy-washy “artsy” stuff or that it isn’t important or useful.

In fact, graphic design is a mix of clear and scientifically sound principles and æsthetic judgement. An understanding of the underlying principles will help you to communicate more effectively. You can use these principles when you build dialog boxes and program windows as well as when you make Web pages or printed designs.

Introduction

Programmers and engineers are creative people, working on solving complex problems in original ways, but that creativity is often not turned towards typography, graphic design and layout. This paper is intended to introduce some of the principles of typography in such a way that programmers can quickly discover the most important principles that they can then apply themselves. Rather then focus on the tools of the trade – font formats, kerning, colour separation, rules of punctuation – I have chosen to focus on the larger picture of why it’s useful to know about typography and graphic design, and in particular to relate this to the human vision system so that you can see it’s based on real and solid principles.

The rules of graphic design are, of course, made to be broken, often with stunning success. But before you can break the rules you have to know what they are.

The following sections each describe a single basic principle. To aid memory I have given them names whose first letters taken together form a word.

The interested reader is also encouraged to see the *further reading* section at the end of this paper, where a short annotated bibliography describes some resources for learning more.

Grouping

The human eye tends to see things that are next to each other as related to one another, and things that are far apart as unrelated.

In Illustration 1, the viewer may perceive two columns of shapes, one double and one single, or may see one group of six shapes and one group of three, but will be unlikely to respond, “I see six circles and three squares.”

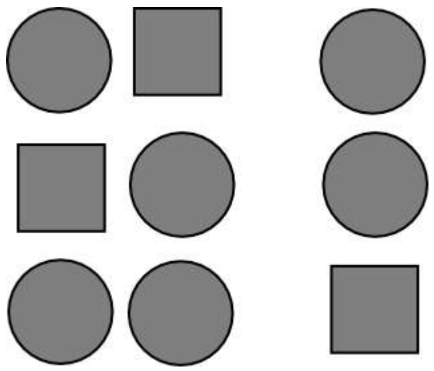


Illustration 1: Nine shapes, or two separate groups of shapes?

This effect is the result of biological vision processing, which tries to make sense of the world by clustering visual data into objects.

As a practical result, put labels on dialog controls near to the corresponding control, and not near to other labels; put headings nearer to the text to which they refer than to preceding or other text. You see Web browsers often giving equal space before and after a heading, when there should be more space above the heading than below.

The way we read text involves parallel processing, with one part of the brain recognising overall word shapes and another part controlling the motion of the eye from word to word. From the principle of *grouping* you won’t be surprised that the word spacing has to be less than the line spacing, because otherwise the eye tends to jump between lines when reading. A good default for word spacing is the width of a lower case “i” in most cases. Fonts usually contain a default value for this, too.

The longer the line of text, the longer it takes the eye to move back to the start, and hence the more line spacing you need. Fonts with a small *x-height* tend to need more line spacing too. A consequence is that whenever you let a user choose a font, a font size, a line length or word spacing, you must also let them choose line spacing. Some TrueType and OpenType fonts contain a default value, but this is almost always too small for continuous text. A good default is 120% of the font size, so that a font that says it’s ten point should have baselines twelve points apart, unless the lines are very long or very short, or the font has unusual proportions.

Since the ascenders and descenders of lower case Latin script make distinctive word shapes, you also would expect that lower case is easier to read, and experiments have confirmed this. This is why the US cigarette companies print health warnings in upper case, by the way.

Uniformity

Just as the eye perceives relationships based upon proximity, it also notices things that are treated differently from their peers, things that "stand out." If you use a circular bullet to introduce list items in all places except one, where you use a star, people will notice and wonder why. They will assume significance to variations in style even if no significance was intended.

This means, of course, that you should treat similar things in similar ways. If you use a small arrow to indicate the presence of a pop-up "context" menu in one place, do it everywhere, and be aware that users will then be surprised if there's a context menu somewhere that doesn't have the visible symbol. A visible symbol whose purpose is to provide people with a clue that they can click on something is called an "affordance." An example in the every-day world might be the row of ridges showing where you should push and slide in order to open a battery compartment.

This principle can be combined with that of *grouping* to say that regular layouts are to be preferred. However, the human vision system changes with age, and children tend to deal better with irregular layouts. If you play memory card-games with children, you can bias the game in favour of the children by using an irregular layout of cards, or in favour of the adult with a regular arrangement in columns and rows. Software for children needs to be designed with this in mind, as do printed books.

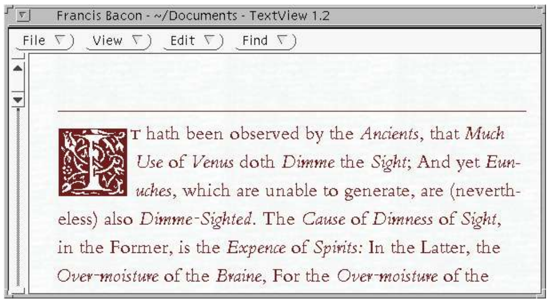
From a programmer's perspective, you can encourage uniform treatment of objects, whether in printed designs or of widgets in windows, by providing high-level objects that automate the repetition. For example, a window control with a label can usually be considered as a single object, and then the spacing between the control and object can be automated.

Alignment

The human eye has a conscious resolution of approximately 10^{4} radians. It is not difficult to construct experiments to measure this. In practice, this means that a misalignment of under a tenth of a point (1/720th of an inch, or about 1/2th of a millimetre, is *consciously* detected by the human eye. A person may be *subconsciously* aware of even smaller misalignments, of course.

This means that if two things almost line up, the design will generally be made stronger if they *exactly* line up with each other. For text, you should normally align baselines, the imaginary line upon which Western characters sit. For non-Latinate scripts there are other conventional alignment points. If two adjacent words are in different size fonts, the baselines must still align perfectly. Approximate alignment is very noticeable.

If you have a text field where users can enter information, and an associated label next to it, the baselines of the text in the label and in the text field should align perfectly. If you have multiple text entry boxes one above the other, align the left-hand edges (if Western) of the boxes. Since you need the labels as near as possible to their boxes, this means you end up right-aligning the text of the labels. This means it's harder to skim down the list of labels, but if you have a dialog box with several hundred text entry boxes you should reconsider the design altogether! With fewer than ten items, the right alignment gives a huge improvement in appearance for little or no loss in speed, and with more than ten items, the loss in ability to skim down is balanced by the fact users are less likely to lose their place within the list when filling in data.

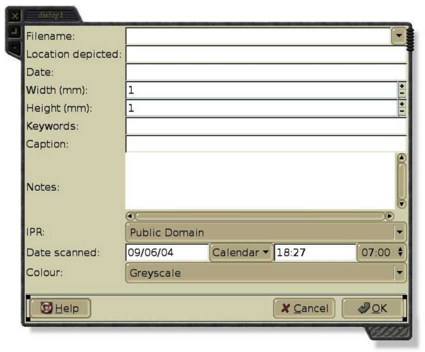


IT hath been observed by the *Ancients*, that *Much Use* of *Venus* doth *Dimme* the *Sight*; And yet *Eunuches*, which are unable to generate, are (nevertheless) also *Dimme-Sighted*. The *Cause* of *Dimness* of *Sight*, in the Former, is the *Expence* of *Spirits*: In the Latter, the *Over-moisture* of the *Braine*, For the *Over-moisture* of the

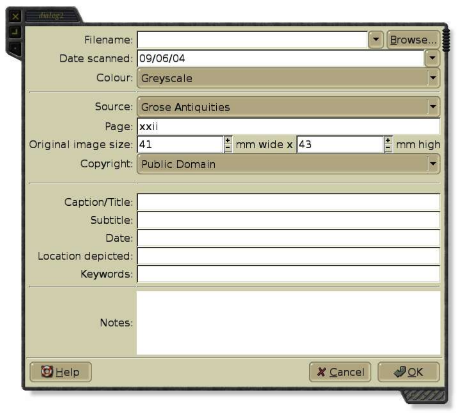
Illustration 2: Example of a drop cap, using a decorative initial Note the alignment with the baseline of the third line of text, and with the cap height in the first line.

Designs in which things are *precisely* aligned tend to look slick. Designs in which things are *almost* aligned tend to look sloppy. Illustration 2 shows a drop capital done right. Notice how the top of the large initial aligns with the top of the first line of text, and the baseline of the initial (the feet, if you will) aligns with the baseline of the third line of text. Getting this right involves careful control of the line spacing, and then taking the cap height of the first line of text into account. It's not simply a matter of multiplying the text size by the number of lines.

Illustration 3 shows a dialog box done in two ways: the same information is made available in both cases, but one looks


dialog1

Filename:	
Location depicted:	
Date:	
Width (mm):	1
Height (mm):	1
Keywords:	
Caption:	
Notes:	
IPR:	Public Domain
Date scanned:	09/06/04 Calendar 18:27 07:00
Colour:	Greyscale


dialog2

Filename:	Browse...
Date scanned:	09/06/04
Colour:	Greyscale
Source:	Grose Antiquities
Page:	xxii
Original image size:	41 mm wide x 43 mm high
Copyright:	Public Domain
Caption/Title:	
Subtitle:	
Date:	
Location depicted:	
Keywords:	
Notes:	

Illustration 3: Redesigned dialog: labels are close to the corresponding controls. Fields have also been grouped to bring out their meaning.

slicker than the other, and less distracting. People can interact with it without noticing its layout, which is a hallmark of clear and effective design. It could easily be improved upon, and is not HIGified, but I hope it's clear that the principles of alignment and grouping have already made the dialog more effective.

Differentiation

Just as the principle of *uniformity* says that if two things are the same, they should look the same, the principle of *differentiation* says that if you want two things to be different, you should make them clearly different. This principle is moderated by the principle of *elegance*, which will be discussed in the next section.

You can differentiate using colour, size, boldness, style, position, and on the screen with animation; a text reader can use pitch, volume, speed and even voice gender. Do not, however, overdo things. If a page of text contains a single word in a bold font and a larger size, the reader's eye will immediately be drawn to it. If it's a heading this might not be a bad thing, as it helps the reader to find the start of a section. If it's a word in the ***middle*** of a sentence, you don't really want the reader to notice how you've differentiated it from the test of the text, but only to notice that it's different. The user shouldn't read all the emphasised words on a page first, and then all the keywords (like *uniformity*) and then the rest of the text. They should notice the emphasised words only when they reach them in a sentence. For these reasons, use *italic* to stress a word. In Gemany it's common to use l e t t e r s p a c i n g for emphasis, perhaps because this slows down the reader, but in other countries this is not usual and will most likely be perceived as a printing error.

However you choose to differentiate design elements, try to do so uniformly (of course) so that the user isn't distracted. Our ability to perceive differences in style, in colour, in alignment, are very finely-tuned, and we may subconsciously notice a difference that we aren't consciously aware of.

Elegance

This principle is also sometimes called *proportion* or *scale* or even *fitness*. If you take a large sheet of paper (A3, say) and put in 7pt (so small it's hard to read) the words "White Socks Not Allowed Here", and pin the sign to your office door, people will be confused. Similarly, overly large text can startle people. The warning labels on cigarette packets in the USA use small text, often smaller than the brand name, and this subconsciously communicates a degree of unimportance. The labels in Canada and the UK are much larger, communicating a sense of urgency.

A sense of scale, then, sounds very subjective. There's certainly a large cultural part of it, but there are constraints on scales and proportions that appear to be innate to the human vision system across all cultures.

If you draw rectangles of varying proportions and ask people which ones are most pleasing, you might expect pretty random answers, but in fact there are fairly constant answers world-wide. People prefer rectangles in the Golden Proportion to all others. Next to that they prefer ratios with simple whole numbers such as 3:4, so that a piece of paper nine centimetres on one side and twelve on the other will be found more pleasing to most people than one that's nine centimetres on one side and eleven on the other: 9:12 is the same as 3:4, and is a harmony, but 9:11 is a dissonance. A few other proportions are also found pleasing, although less than the golden ratio. One in particular is 1?2, and this ratio is the basis for the ISO metric paper sizes used in most countries outside North America.

If you have chosen paper of a pleasing size, you then need to realize that the outline of the body of text on that paper also makes a rectangle which should, where possible, be in a pleasing ratio, and ideally the same ratio as that of the paper. This constrains the corners of the text area to be on the leading diagonal of the page, a fact known to mediæval scribes but forgotten when typography passed into the hands of engineers who did not have the formal training given to apprentices. This does not reflect badly on the engineers, but rather on the secrecy of the guild system used at the time to control knowledge and power. Illustration 4 shows the diagram of page proportions, which was rediscovered by Jan Tschichold by studying construction marks still visible on old books.

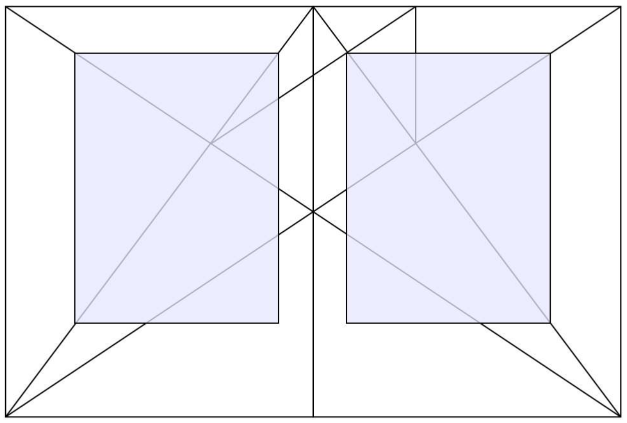


Illustration 4: Margins in this duoble page are in the ration 1:1:2:3, with a division into ninths. [Tsichichold p. 36ff]

I have seen very few windowing systems that consider proportions of windows or dialog boxes at all. This is unfortunate, as it's clear that a mixture of mismatched proportions creates needless tension and discomfort on the part of most users.

Just as page proportions are important, so is the use of proportion in other aspects of work. One should try to aim for the minimum amount of *contrast* necessary to achieve a goal. Consider, for example, chapter headings. If you have a single level of heading, so that a document contains headings and text, and if the headings are short, there may be no need to do more than put them in a bold font and have extra space before them (not after, see*grouping*). If the headings are numbered, you might not need the boldness. If you have headings and sub-headings, and you want readers to be able to skip quickly from section to section, skipping over sub-headings, you'll need to use two forms of differentiation. You might make the chapter heading slightly larger (e.g. 14pt instead of 12pt) with an extra line or two of space before it, for example, or you might use bold for headings and italic (not bold-italic!) for sub-headings. A common error is to make headings too large. Mosaic and Netscape Navigator both had this problem, and for this reason, before the advent of CSS, graphic designers avoided the **h1** element entirely. It was out of proportion, too big for the screen, and entirely unusable.

I've lumped *proportion* and *scale* under *Elegance*, because in an elegant design the proportion and size of all the elements is such that all elements relate to each other by the degree of their overall importance. In other words, you can make a design that is both slick and elegant by paying attention to all the principles described in this paper. A*beautiful* design also requires art, which is another matter entirely, of course.

Colour

There are two main senses in which the term *colour* is used in typography and graphic design. The obvious meaning (*chromatic colour*) refers to hue, of course, such as yellow, red, chartreuse and puce. The other meaning is found in the term *typographic colour*, and this refers to how light or dark a block of text appears. I'll take them in reverse order.

Typographic Colour is affected by word spacing, letter spacing, kerning (which should always be turned on by default, Microsoft Word and OpenOffice notwithstanding), line spacing and of course choice of font. In well-designed pages the colour is even from one page to the next. There are no obvious unintended light or dark spots to distract the reader, and if you flip through a book quickly you don't see some light and some dark pages where the text was uneven. Because the eye is good at detecting differences and patterns, this evenness is the result of careful hard work. The*Further Reading* section mentions a couple of books that will help you if you need to learn more about this.

Chromatic Colour in general is often used on computer screens, but it should not normally be used only for decoration. Use it to convey meaning, but be aware that people perceive colour differently. Some people can't distinguish red from black, others can't distinguish colours from each other at all. There are also cultural differences. In China, red is used to indicate good fortune, so that if a button appears with a red label, people will probably click on it in preference to a green label. In the West, red is more often used to mean danger, and green safety. A user interface that employs colours needs to be customizable both for individuals and for cultural preferences.

When you use colour to *differentiate* elements of a design, remember not to overdo it: one or two yellow lines or bold words can be much more effective than a whole page gaudily blinking. Time Square in New York might be an interesting experience but few people would call it *restful* or say that the lights and sounds did not distract them at all.

Careful, limited use of colour can make a design much more effective.

Summary

The principles of **G**rouping, **U**niformity, **A**lignment, **D**ifferentiation, **E**legance and **C**olour are not arbitrary. Although they all have aesthetic aspects, they are also governed by the nature of human perception. There are many other important principles in typography, but this paper must be presented in only a short session, and in any case the acronym ran out of letters. In her book *The Non-Designer's Design Book*, Robin Williams describes Contrast, Repetition, Alignment and Proximity, which correspond closely to Differentiation, Uniformity, Alignment and Grouping, but the terms I have chosen work slightly better for user interface design. Kevin Mullet and Darrel Sano in *Designing Visual Interfaces* also talk about proportion and scale, which I have put together under the result of applying them; Elegance.

Above all else, I hope that I've given readers an insight into the way that typography and graphic design – the user interface of text and idea – are based on scientific principles that can be fairly easily explained, understood and applied. It should be admitted that the art and skill lies not only in how to apply the rules, but in when and how to break them.

Further Reading

1. Dowding, Geoffrey, *Finer Points in the Spacing & Arrangement of Type*, Hartley & Marks Publishers Inc., 1966, reprinted in 1995 with a foreword by Crispin Elsted. This excellent volume has many clear examples, together with rules of thumb such as `A round and generously formed letter like Baskerville can be set to a wider measure [line length] than a condensed face like Fournier.'

2. Williams, Robin, *The Non-Designer's Design Book*, Peachpit Press, 1994. Robin uses a different mnemonic than I chose: Contrast, Repetition, Alignment, Proximity, with a less Gnomish but more memorable acronym. Her book is a definite classic: it's also short and easy to read. Anyone involved in layout, typography or graphic design at any level is likely to have a copy of this book.

3. Bringhurst, Robert, *The Elements of Typographic Style*, Hartley & Marks 1999 (Second edition). Robert's writing is steeped in folklore and anecdotes of printing, but if you decide you're interested in typography this is a splendid and useful book to get. It's more advanced than Robin Williams' *The Non-Designer's Design Book*, and if you only get one book, get Robin's. But if you get two, let this be your second.

4. Tschichold, Jan, *The Form of the Book: Essays on the Morailty of Good Design*, Hartley & Marks 1981. An English translation (the original is in German) of a collection of essays on book design. The diagram showing page proportions is from this book. I don't expect any except the utterly fascinated to explore this book, but I list it because I used his diagram and mentioned his discovery of page ratios. Jan was one of the foremost graphic designers and typographers of the 20th century.

Custom widgets using GtkCairo

Øyvind Kolås
Gjøvik University College,
P.O.Box 191
N-2802 Gjøvik, Norway

oeyvindk@hig.no

May 16, 2004

Abstract

Cairo is a vector graphics library featuring hardware acceleration and cross-device output support.

A basic introduction to cairo's vector-based rendering API, and how to make custom widgets rendered with cairo.

1 Introduction

Cairo[4] is a device independent drawing API, based on the PostScript programming model, with support for the PDF 1.4[2] imaging model.

GtkCairo is a cairo[4] widget for The Gimp Toolkit (GTK). GtkCairo provides a canvas for drawing using cairo's vector-based rendering API.

2 Initializing

The code in fig, 1 creates a new GtkCairo widget, and hooks up a redraw handler. The redraw function will contain the drawing code using the cairo API.

```
static void
redraw (GtkCairo *gtkcairo,
        cairo_t  *cr,
        gpointer *data);

{
   GtkWidget *gtkcairo;

   gtkcairo = gtk_cairo_new ();
   g_signal_connect (G_OBJECT (gtkcairo), "redraw",
                     G_CALLBACK (redraw), NULL);

   gtk_widget_set_size_request (gtkcairo, 256, 256);
}
```

Figure 1: Initializing GtkCairo.

```
cairo_move_to (cr, 0.5, 0.1);
cairo_line_to (cr, 0.9, 0.9);
cairo_rel_line_to (cr, -0.4, 0.0);
cairo_curve_to (cr, 0.2, 0.9, 0.2, 0.5, 0.5, 0.5);

cairo_stroke (cr);
```

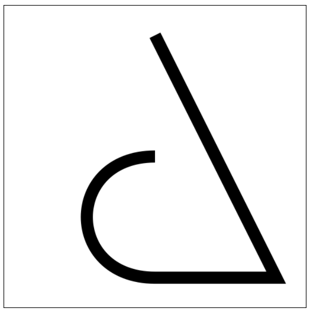


Figure 2: Making a simple path.

```
double x0=0.1, y0=0.5,
       x1=0.4, y1=0.9,
       x2=0.6, y2=0.1,
       x3=0.9, y3=0.5;
cairo_move_to (cr, x0, y0);
cairo_curve_to (cr, x1, y1, x2, y2, x3, y3);
cairo_stroke (cr);

cairo_set_rgb_color (cr, 1,0.2,0.2);
cairo_set_alpha (cr, 0.6);
cairo_set_line_width (cr, 0.03);
cairo_move_to (cr,x0,y0); cairo_line_to (cr,x1,y1);
cairo_move_to (cr,x2,y2); cairo_line_to (cr,x3,y3);
cairo_stroke (cr);
```

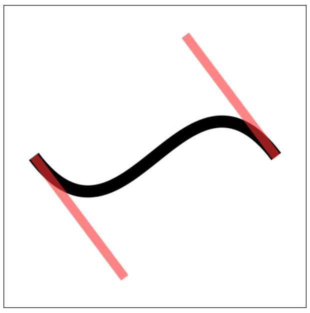


Figure 3: Control points of a bezier spline.

3 Coordinate system

Cairo's origin is located in the upper left corner of the drawing surface with the X axis to the right and Y axis downwards. The default mapping between *device space* and *user space* is a one to one; making it easy to render pixel-aligned lines and rectangles. The code snippets in this paper uses a *user space* equal to the unit square `(0,0)-(1,1)`, fig. 7 shows how this is achieved using GtkCairo.

4 Paths

A path is a collection of, possibly disjoint, lines and areas describing shapes. Fig. 2 illustrates the process of constructing a simple path. The first function moves the *current point*, after this the path is incrementally constructed from segments, the first two are straight lines, one specifying it's end point in absolute coordinates, the other with coordinates relative to the current point.

4.1 Bezier splines

The final segment in fig. 2 is a bezier spline, `cairo_curve_to` is passed three coordinate pairs as parameters, the first two are the control points controlling the curvature and the third is the end point of the curve.

```
cairo_set_line_width (cr, 0.12);
cairo_set_line_cap  (cr, CAIRO_LINE_CAP_BUTT); /* default */
cairo_move_to (cr, 0.25, 0.2); cairo_line_to (cr, 0.25, 0.8);
cairo_stroke (cr);
cairo_set_line_cap  (cr, CAIRO_LINE_CAP_ROUND);
cairo_move_to (cr, 0.5, 0.2); cairo_line_to (cr, 0.5, 0.8);
cairo_stroke (cr);
cairo_set_line_cap  (cr, CAIRO_LINE_CAP_SQUARE);
cairo_move_to (cr, 0.75, 0.2); cairo_line_to (cr, 0.75, 0.8);
cairo_stroke (cr);
cairo_move_to (cr, 0.25, 0.2); cairo_line_to (cr, 0.25, 0.8);
cairo_move_to (cr, 0.5, 0.2);  cairo_line_to (cr, 0.5, 0.8);
cairo_move_to (cr, 0.75, 0.2); cairo_line_to (cr, 0.75, 0.8);
cairo_set_rgb_color (cr, 1,1,1);
cairo_set_line_width (cr, 0.01);
cairo_stroke (cr);
```

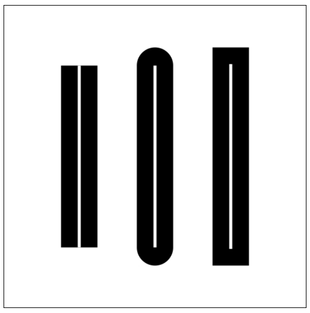


Figure 4: Available line caps.

5 State

Cairo's rendering API is stateful, it keeps track of the current path, mapping of coordinate system, font information etc. State manipulation functions don't draw and drawing functions don't change state.

5.1 Graphics state

The *graphics state*, keeps track of the information about how to render graphic primitives. The functions for doing the most basic graphics state manipulations are:

`cairo_set_rgb_color (cairo_t *cr, double red, double green, double blue)` sets the current color using values in the range 0.0-1.0.

`gtk_cairo_set_gdk_color (cairo_t *cr, GdkColor *color)` sets the current color from a *GdkColor*. This allows setting colors from the current style[1].

`cairo_set_alpha (cairo_t *cr, double alpha)` sets the current transparency of the current color, the valid range 0.0 (transparent) to 1.0 (opaque)

`cairo_set_line_width (cairo_t *cr, double linewidth)` sets the current line width in *user space* units.

`cairo_set_line_cap (cairo_t *cr, cairo_line_cap_t line_cap)` sets the cap drawn at ends of stroked segments, see fig. 4

`cairo_set_line_join (cairo_t *cr, cairo_line_join_t line_join)` sets the style of which segments of a path are joined see fig. 5

5.2 Saving and restoring state

Cairo maintains a stack of states, issuing `cairo_save` saves the current state, while `cairo_restore` restores the previously saved state, the number of restores must match the number of saves.

[1]`gtk_cairo_set_gdk_color (cr, &(GTK_WIDGET (gtkcairo)->style->base[GTK_STATE_NORMAL]).`

```
cairo_set_line_width (cr, 0.12);
cairo_move_to (cr, 0.3, 0.3);
cairo_rel_line_to (cr, 0.2, -0.2);
cairo_rel_line_to (cr, 0.2, 0.2);
cairo_set_line_join (cr, CAIRO_LINE_JOIN_MITER); /* default */
cairo_stroke (cr);

cairo_move_to (cr, 0.3, 0.6);
cairo_rel_line_to (cr, 0.2, -0.2);
cairo_rel_line_to (cr, 0.2, 0.2);
cairo_set_line_join (cr, CAIRO_LINE_JOIN_BEVEL);
cairo_stroke (cr);

cairo_move_to (cr, 0.3, 0.9);
cairo_rel_line_to (cr, 0.2, -0.2);
cairo_rel_line_to (cr, 0.2, 0.2);
cairo_set_line_join (cr, CAIRO_LINE_JOIN_ROUND);
cairo_stroke (cr);
```

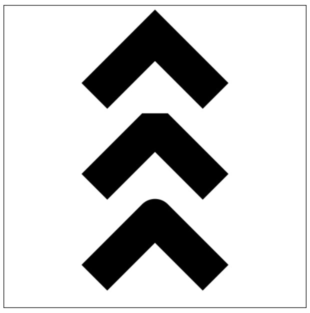


Figure 5: Available line joins.

```
cairo_move_to (cr, 0.5, 0.1);
cairo_line_to (cr, 0.9, 0.9);
cairo_rel_line_to (cr, -0.4, 0.0);
cairo_curve_to (cr, 0.2, 0.9, 0.2, 0.5, 0.5, 0.5);

cairo_save (cr);
    cairo_set_rgb_color (cr, 0, 0, 1);
    cairo_fill (cr);
cairo_restore (cr);

cairo_close_path (cr);
cairo_stroke (cr);
```

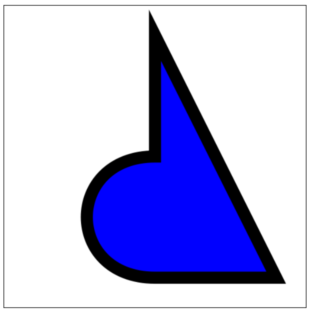


Figure 6: Filling and stroking a path.

6 Filling and stroking

Fill and stroke operations clear the *current path*, by saving the state before filling, and restoring afterwards the path is preserved and is available for stroking.

This is done in Fig. 6, note that even though the path was not closed a solid shape is drawn. The `cairo_close_path` function makes a closed shape of the segments since the last `cairo_move_to` or path construction started.

7 Affine Transforms

The current transform matrix *CTM* expresses the current mapping between *device space* and *user space*.

`cairo_scale (cairo_t *cr, double sx, double sy)` changes the scale factor, effectively zooming in/out around the origin.

`cairo_rotate (cairo_t *cr, double angle)` rotates the user space the specified amount of radians around the origin.

`cairo_translate (cairo_t *cr, double tx, double ty)` sets the origin to a new point, specified in current user space coordinates.

Setting the user space to the unit square for GtkCairo can be achieved by using the code in fig. 7.

```
static void
redraw (GtkCairo *gtkcairo,
        cairo_t  *cr,
        gpointer *data)
{
   GtkWidget *widget = GTK_WIDGET (gtkcairo);

   cairo_save (cr);
   cairo_scale (cr, widget->allocation.width,
                    widget->allocation.height);

   /* actual drawing code */

   cairo_restore (cr);
}
```

Figure 7: Changing the Current Transform Matrix (CTM).

```
int              w,h,stride;
char            *buffer;
cairo_surface_t *image;

buffer = read_png_argb32 (
              "data/romedalen.png", &w,&h, &stride);
image  = cairo_surface_create_for_image (
              buffer, CAIRO_FORMAT_ARGB32, w,h, stride);

cairo_translate (cr, 0.5, 0.5);
cairo_rotate (cr, 45*PI/180);
cairo_scale  (cr, 1.0/w, 1.0/h);
cairo_translate (cr, -0.5*w, -0.5*h);

cairo_move_to (cr, 0,0);
cairo_show_surface (cr, image, w, h);
cairo_surface_destroy (image);
free (buffer);
```

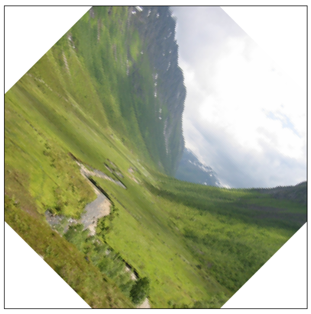


Figure 8: Image transformed through CTM.

8 Images

Cairo memory surfaces are images, one surface can be painted into another by using `cairo_show_surface`. The image is transformed through the *CTM*. Fig. 8 shows loading of a PNG image and displaying it rotated 45 degrees.

9 Gradients

Cairo supports patterns for non solid filling and stroking, a pattern can either be created from other cairo surfaces, or be a linear or radial gradient. A source matrix is associated with each pattern governing the mapping between to user space.

In fig. 9 usage of linear and radial gradients is demonstrated. The linear gradient is specified by start and endpoints, whilst the radial gradient is specified by coordinates and radius of an inner, and outer circle. Using the `cairo_pattern_add_color_stop` function, the colors to be used between start (0.0) and end (1.0) is specified.

```
cairo_pattern_t *pat;
pat = cairo_pattern_create_linear (0.0, 0.0,  0.0, 1.0);
cairo_pattern_add_color_stop (pat, 1, 0, 0, 0, 1);
cairo_pattern_add_color_stop (pat, 0, 1, 1, 1, 1);
cairo_rectangle (cr, 0,0,1,1);
cairo_set_pattern (cr, pat);
cairo_fill (cr);

cairo_pattern_destroy (pat);
pat = cairo_pattern_create_radial (0.45, 0.4, 0.1,
                                   0.4,  0.4, 0.5);
cairo_pattern_add_color_stop (pat, 0, 1, 1, 1, 1);
cairo_pattern_add_color_stop (pat, 1, 0, 0, 0, 1);
cairo_set_pattern (cr, pat);
cairo_arc (cr, 0.5, 0.5, 0.3, 0, 2*PI);
cairo_fill (cr);
cairo_pattern_destroy (pat);
```

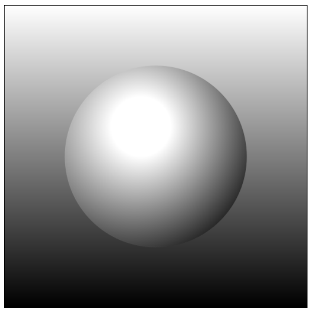


Figure 9: Linear and radial gradient.

```
cairo_select_font (cr, "Sans", CAIRO_FONT_SLANT_NORMAL,
                               CAIRO_FONT_WEIGHT_BOLD);
cairo_scale_font (cr, 0.35);

cairo_move_to (cr, 0.04, 0.53);
cairo_show_text (cr, "Hello");

cairo_move_to (cr, 0.27, 0.65);
cairo_text_path (cr, "void");
cairo_save (cr);
    cairo_set_rgb_color (cr, 0.5,0.5,1);
    cairo_fill (cr);
cairo_restore (cr);
cairo_set_line_width (cr, 0.01);
cairo_stroke (cr);
```

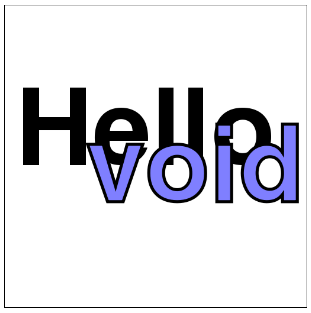


Figure 10: Normal and outlined text.

10 Text

Cairo includes a simple “Toy” text API, that allows displaying UTF8 strings. When a planned pango integration is complete, cairo will be able to operate on an array of glyphs positioned by pango.

`cairo_select_font (cairo_t *cr, `*`family, slant, weight`*`)` selects a font face

`cairo_scale_font (cairo_t *cr, double scale)` scales the current font, initially a fonts height is 1 user space unit.

`cairo_show_text (cairo_t *cr, char *utf8)` draws the supplied UTF8 string using the current color and alpha values.

`cairo_text_path (cairo_t *cr, char *utf8)` adds the outline of the supplied UTF8 string to the current path.

11 Updating the canvas

Invoking `gtk_widget_queue_draw (widget)` on your GtkCairo widget invalidates its display. Multiple calls to `gtk_widget_queue_draw` are collapsed, and only a single redraw will be invoked.

```
static gboolean
event_press (GtkWidget       *widget,
             GdkEventButton  *bev,
             gpointer         user_data);

{
    /* continued from Fig.1 */
    ..
    gtk_widget_add_events (gtkcairo, GDK_BUTTON_PRESS_MASK);
    g_signal_connect (G_OBJECT (gtkcairo), "button_press_event",
                      G_CALLBACK (event_press), NULL);
}
```

Figure 11: Registering for user events.

12 Handling user events

Cairo itself doesn't provide any means for user interaction, but since GtkCairo derives from GtkWidget, all the standard methods registered for a widget apply. You should add the eventmask, and handlers for events you want to intercept as in figure 11, when creating your widget.

Refer to the GTK+ Reference Manual[1], and the section on custom widgets in the GTK+ tutorial[3] for further information.

12.1 Insideness checking

You can use cairo to check whether a coordinate is inside the stroke or fill of the current path using the functions `int cairo_in_stroke (cairo_t *cr, double x, double y)` and `int cairo_in_fill (cairo_t *cr, double x, double y)`.

To transform between *device space* (events produced by GTK+) and cairo's *user space* you can use the function `cairo_inverse_transform_point (cairo_t *cr, double *x, double *y)` which transforms the values stored at the addresses pointed to by the arguments.

References

[1] Inc. Free Software Foundation. GTK+ Reference Manual, May 2004. `http://developer.gnome.org/doc/API/2.0/gtk/`.

[2] Adobe Systems Inc., editor. *PDF Reference: Version 1.4*. Addison-Wesley, 3rd edition, 2001.

[3] Ian Main Tony Gale and the GTK team. GTK+ 2.0 Tutorial, May 2004. `http://www.gtk.org/tutorial/`.

[4] Carl Worth and the Cairo development team. Cairo: A Vector Graphics Library, 2003. `http://www.cairographics.org`.

BOF: Molos – Master on libre / free / open source software

Janis Gailis <Janis.Gailis@hia.no>
Jesus Jesus M. Gonzalez-Barahona <jgb@gsyc.escet.urjc.es>

Molos is a master's programme to be offered jointly by several universities across Europe.

The subject of the master's will be "libre (free, open source) software", a field which has recieved little attention in higher education courses, but which is attracting increasing attention all over the world. The design of the curriculum not only provides the needed education on this field, but also explores ways of translating into education some practices that the libre software community has long demonstrated in the areas of quality assurance, collaboration among geographically distributed groups, creation of texts by communities of authors, etc. The course development also explores new directions in mixed learning practices, both in-place and distance, combining intensive teaching of both theoretical and practical topics.

A solid group of universities has been assembled to develop the master's programme, with a multidisciplinary team of people already active in the libre software community, with a good knowledge of it, and with extensive pedagogical experience and an interest in education about libre software topics.

A proposal of a curriculum will be presented and the following things could be interesting to discuss:

- the general need for a specific open source related education
- multidisciplinary, - how many and what kind of tracks?
- GNOME's motivation and possible contribution to the implementation of Molos
- accreditation issues for different countries
- open source projects, - initialization, use in educational situations and as a tool for co-operation

HAL and GNOME

David Zeuthen

david@fubar.dk

HAL is a hardware abstraction layer and aims to provide a live tree of devices present in the system at any point in time. Each device has a number of properties defined that can stem from several sources including the hardware itself or from device information files matching a specific device. Desktop environments, like GNOME, can communicate with the HAL through a system-wide message bus for querying of devices and asynchronous notifications when device properties change and devices are inserted/removed. This paper discusses the HAL and the current level of integration of the HAL into the GNOME desktop and developer platform.

Table of Contents

Introduction

Notwithstanding the wide availability of device drivers for free software operating systems, the way a user interacts with devices is often perceived as sub optimal. On many occacions the user needs to resort to the command line and OS-specific configuration files when dealing with hardware and there are only limited graphical configuration tools available. One of the prime reasons this is, stems from the fact that most desktop environment are cross platform so little can be assumed about the underlying operating system.

This paper discusses one effort, the Hardware Abstraction Layer, HAL, that specifically attempts to bridge the gap between the operating system and desktop environments such as GNOME or KDE. This paper focuses primarily on the HAL but Project Utopia, an umbrella project to fully integrate the GNOME desktop, is also briefly discussed.

Abstraction of Hardware

The Hardware Abstraction Layer, HAL, provides a system-wide daemon that maintains a live list of the various devices installed in the system. It is designed to be operating system and desktop environment independent.

Information exported by the HAL

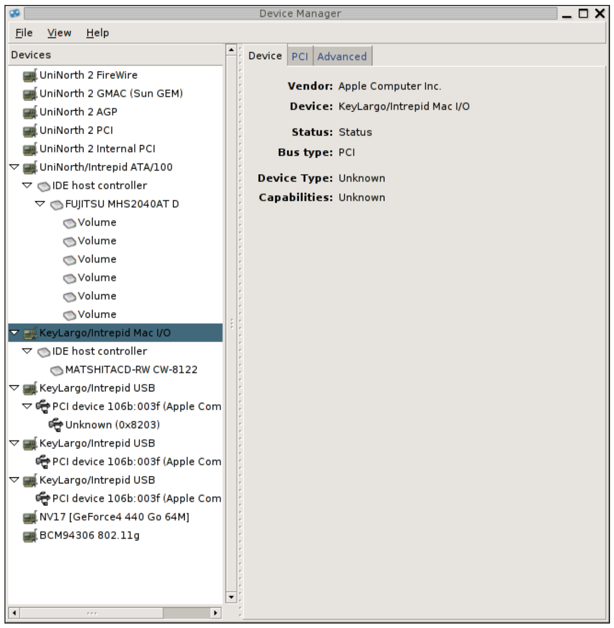
One of the foremost duties of the HAL is to ensure that as much information as possible about a device is present and current - this includes, as a bare minimum, enough information to use the device, but it may also include other information such as what kind of media a card reader uses.

This information can stem from several sources - some is read from operating system or the hardware itself, other information is read from vendor provided files and some may be read from device information files. The HAL merges all this information together as typed well-defined properties on the device. All the properties exported by the HAL are clearly documented in the HAL specification.

Bus and Class Devices

The HAL exports the devices in the system as a set of "Device Objects" where each object represent a bus device, e.g. a USB, PCI device. In addition, HAL also export software-only abstractions of bus devices from the kernel such as SCSI-glue for USB or IEEE1394 storage devices. Devices are identified by a Unique Device Identifier, UDI. Aside from being unique the UDI is also stable, e.g. it doesn’t change if the device is plugged into a different port and if the device is removed and subsequently plugged in the UDI will be the same. [1]

Devices always have a parent property, if applicable, so it is possible to draw a tree as in Figure 1 that shows how devices are connected. Since bus device types are inherently operating system independent, this means that such a tree should look the same on any operating system where the HAL is implemented.



UniNorth 2 FireWire
UniNorth 2 GMAC (Sun GEM)
UniNorth 2 AGP
UniNorth 2 PCI
UniNorth 2 Internal PCI
UniNorth/Intrepid ATA/100
IDE host controller
FUJITSU MHS2040AT D
Volume
Volume
Volume
Volume
Volume
Volume
KeyLargo/Intrepid Mac I/O
IDE host controller
MATSHITACD-RW CW-8122
KeyLargo/Intrepid USB
PCI device 106b:003f (Apple Com
Unknown (0x8203)
KeyLargo/Intrepid USB
PCI device 106b:003f (Apple Com
KeyLargo/Intrepid USB
PCI device 106b:003f (Apple Com
NV17 [GeForce4 440 Go 64M]
BCM94306 802.11g

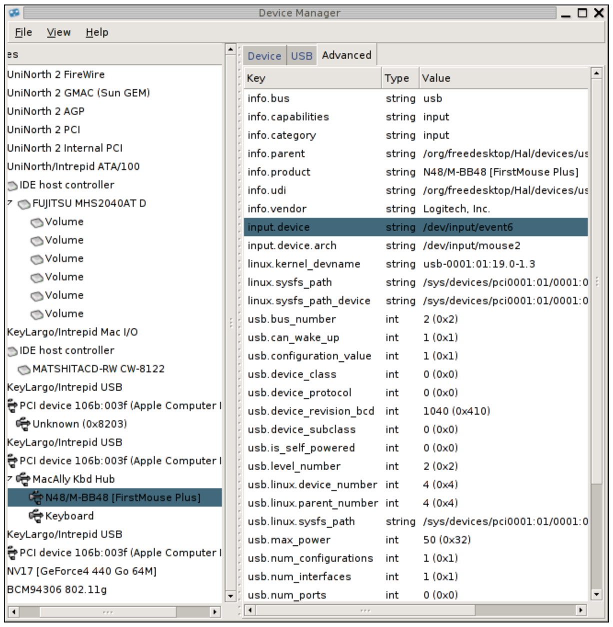
Vendor: Apple Computer Inc.
Device: KeyLargo/Intrepid Mac I/O
Status: Status
Bus type: PCI
Device Type: Unknown
Capabilities: Unknown

Figure 1. Device Tree

Devices may vary between manufacturers, e.g two mice often perform essentially the same function - allowing the user to navigate and press a button. On modern operating systems this is abstracted in both kernel- and user-space drives by using the term "class device". This allow applications to just think in terms of "mouse" instead of having to worry if the mouse is connected to a USB port or PS/2 port. It is in general possible, though of little interest, for the application to determine what bus device the class device belongs to.

Some devices may be multi-function, e.g. there may be several class devices that maps to a bus device. Indeed such devices exist, for example a set of USB speakers may provide both an audio class device and a input class device, the latter for applications to react on the user adjusting e.g. speaker volume.

The HAL is concerned with class devices - after all this is what applications are interested in. This is simply achieved by just merging information on the the bus device the class device stems from. In Figure 2 the properties for the an input device is shown; for example the special device file an application use to interact with the device is shown as input.device.



UniNorth 2 FireWire
UniNorth 2 GMAC (Sun GEM)
UniNorth 2 AGP
UniNorth 2 PCI
UniNorth 2 Internal PCI
UniNorth/Intrepid ATA/100
IDE host controller
FUJITSU MHS2040AT D
Volume
Volume
Volume
Volume
Volume
Volume
KeyLargo/Intrepid Mac I/O
IDE host controller
MATSHITACD-RW CW-8122
KeyLargo/Intrepid USB
PCI device 106b:003f (Apple Computer I
Unknown (0x8203)
KeyLargo/Intrepid USB
PCI device 106b:003f (Apple Computer I
MacAlly Kbd Hub
N48/M-BB48 [FirstMouse Plus]
Keyboard
KeyLargo/Intrepid USB
PCI device 106b:003f (Apple Computer I
NV17 [GeForce4 440 Go 64M]
BCM94306 802.11g

Key	Type	Value
info.bus	string	usb
info.capabilities	string	input
info.category	string	input
info.parent	string	/org/freedesktop/Hal/devices/us
info.product	string	N48/M-BB48 [FirstMouse Plus]
info.udi	string	/org/freedesktop/Hal/devices/us
info.vendor	string	Logitech, Inc.
input.device	string	/dev/input/event6
input.device.arch	string	/dev/input/mouse2
linux.kernel_devname	string	usb-0001:01:19.0-1.3
linux.sysfs_path	string	/sys/devices/pci0001:01/0001:0
linux.sysfs_path_device	string	/sys/devices/pci0001:01/0001:0
usb.bus_number	int	2 (0x2)
usb.can_wake_up	int	1 (0x1)
usb.configuration_value	int	1 (0x1)
usb.device_class	int	0 (0x0)
usb.device_protocol	int	0 (0x0)
usb.device_revision_bcd	int	1040 (0x410)
usb.device_subclass	int	0 (0x0)
usb.is_self_powered	int	0 (0x0)
usb.level_number	int	2 (0x2)
usb.linux.device_number	int	4 (0x4)
usb.linux.parent_number	int	4 (0x4)
usb.linux.sysfs_path	string	/sys/devices/pci0001:01/0001:0
usb.max_power	int	50 (0x32)
usb.num_configurations	int	1 (0x1)
usb.num_interfaces	int	1 (0x1)
usb.num_ports	int	0 (0x0)

Figure 2. Properties for an input device

Device Capabilities

As the class device is really the abstraction that applications use to interface with devices, there needs to be some kind of facility such that the application can search for e.g. all cameras connected to the system. In the HAL this is provided by tagging each device with zero or more so called capabilities - a capability is simply a text string like "storage", "printer", "camera" and so forth. Some devices though, like a PCI card with USB ports, are not really interesting for desktop applications so they may not have any HAL capabilities at all.

Having a capability doesn't imply that the device has a specific bus type or anything - the only thing it actually means is that the physical device represented by the HAL device with said capability is a printer or camera etc. A real-world example is digital cameras - either a camera works just like a USB mass storage device accessible through a kernel space driver as a block device. Otherwise it uses a proprietary protocol and can be accessed through e.g. the libgphoto camera library. In the HAL capabilities are both determined from the kernel-level drivers and/or from device information files

Device Information Files

A device information file is a simple XML document that includes information that can merged onto a device. Figure 3 shows a document that matches the authors digital camera.

```
<?xml version="1.0" encoding="ISO-8859-1"?> <!-- -*- SGML -*- -->
```

```
<deviceinfo version="0.2">
  <device>
    <match key="info.bus" string="usb">
      <match key="usb.vendor_id" int="0x04a9">
        <match key="usb.product_id" int="0x3052">
          <merge key="info.category" type="string">camera</merge>
          <merge key="info.capabilities" type="string">camera</merge>
          <merge key="info.persistent" type="bool">true</merge>
        </match>
      </match>
    </match>
  </device>
</deviceinfo>
```

Figure 3. Example Device Information File

All device information files are processed when a device have been detected, but just before it is exported. Thus, it can be used to override information extracted from the hardware or operating system. Device information files can be supplied from the operating system vendor, the device vendor or from community web sites.

Using Devices

The philosophy behind the HAL is to identify what a device does through the notion of capabilities - this easily enables the application programmer to search for a camera or a scanner without having to worry about what kind of bus the device is on.

With regards to using the device it would be ideal to have one shared library for every capability in a way that the application programmer can just pass the UDI for the device he wishes to use. [2] For cameras this would mean that the library should handle both USB storage based cameras as well as cameras supported by e.g. libgphoto. This has the advantage that any vendor can submit a patch to the camera capability library and instantly all applications using that library just works with the new device.

In general, it is a bug if the application programmer would have to look at the device himself using e.g. the special device file, since future devices of same capability could be implemented through a user space driver or another kernel driver with different semantics.

Architecture, Portability and Integration

The HAL daemon is only one piece of the solution to "Making hardware just work" - it is designed to fit in a greater architecture as show in Figure 4

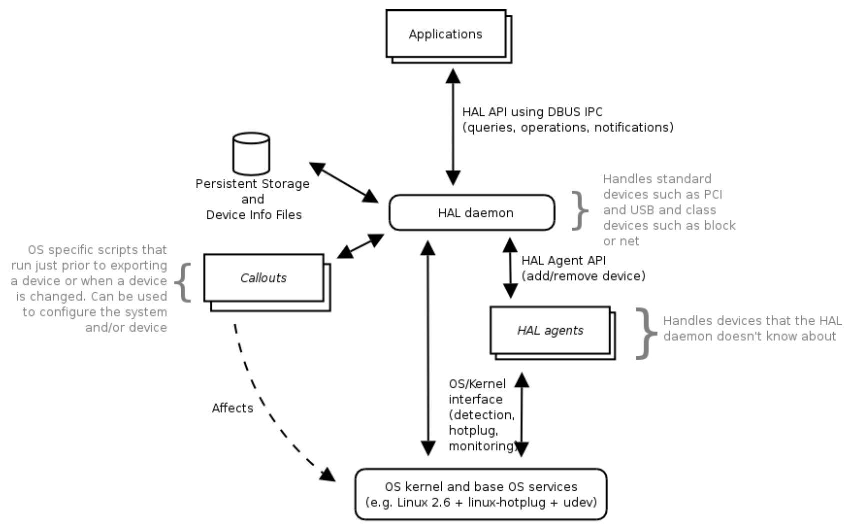

Applications
HAL API using DBUS IPC
(queries, operations, notifications)
Persistent Storage
and
Device Info Files
HAL daemon
Handles standard devices such as PCI and USB and class devices such as block or net
OS specific scripts that run just prior to exporting a device or when a device is changed. Can be used to configure the system and/or device
Callouts
HAL Agent API
(add/remove device)
HAL agents
Handles devices that the HAL daemon doesn't know about
Affects
OS/Kernel interface (detection, hotplug, monitoring)
OS kernel and base OS services
(e.g. Linux 2.6 + linux-hotplug + udev)

Figure 4. HAL architecture

Highlights:

- The HAL daemon maintains a list of devices in the system.
- Information may be merged from device information files.
- Callouts are invoked when a device is added, removed or properties on the device change. Callouts can perform system-wide configuration of the device or the surrounding system and it also be used to merge even more information onto the device. Only OS vendors and third party application are expected to use callouts.
- The HAL communicates with desktop level applications through inter process communication using the DBUS framework. This communication entails handling queries from the application and providing notifications. As such, the HAL can be accessed from any application with DBUS bindings. As a convenience, HAL ships a shared library, libhal, to easily interact with the HAL daemon.

Presently the HAL is implemented only on systems with the Linux kernel 2.6, udev and linux-hotplug. The design of the HAL, though, allows for implementation on other operating systems given they have sufficient infrastructure available. It's imporant to realize, that even systems based on the Linux 2.6 kernel can be very different with respect to system configuration. That is why the HAL daemon has hooks for the OS vendor to have system-wide scripts executed when configuration changes. For example, scripts can be run to update the system when a storage volume is added or removed (e.g. to update the /etc/fstab file), or when a network cable have been inserted or removed (to configure the networking stack).

Project Utopia

Project Utopia is an effort to fully integrate the GNOME desktop in order to "Make hardware just work". It builds upon HAL for abstracting hardware since GNOME runs on many platforms and isn't tied to Linux. The project is under development but it is anticipated that the way it will be implemented is through enhancing existing GNOME components to use the rich information exported by the HAL as well as providing brand new components. The term "halificate", coined by Robert Love and

Joe Shaw, refers to the process of adapting software to use HAL - normally this can be done in a non-intrusive way, typically as a compile-time option much the same way that e.g. Nautilus can optionally use the file alteration monitor daemon if available

Status of the halification of GNOME

Presently, the following components are halificated, meaning either a patch exist or HAL support is in mainline as a compile-time option:

- *gnome-volume-manager*: g-v-m is a session-wide daemon that enforces policy by mounting and unmounting removable media. The type of the media is also detected (e.g. photos or video) and applications are launched according to the policy defined by the user.
- *libbacon*: This GNOME library uses HAL for optical drive and disc detection.
- *gnome-vfs*: The gnome-vfs-daemon, which provides the foundation for the Nautilus file manager, can use HAL to detect removable storage devices.
- *gnome-cups-manager*: When a new printer is detected the "Add Printer" dialog is launched such that the user can configure the printer.

Future efforts obviously include the halification of more applications and libraries but thoughts about a desktop-wide hardware configuration application for plug and play is also on the horizon.

Conclusion

By using the HAL, desktop environments such as GNOME or KDE can bridge the gap between the desktop and the underlying operating system without substantial reliance on e.g. Linux 2.4 or Linux 2.6 specific code. Operating system vendors can reap the benefits of the open source model by sharing an unified system for handling hardware that is fully integrated into the desktop and finally device vendors will have a straightforward roadmap on how to make their devices work well on free software desktops.

Although HAL is still under heavy development, many important use cases are already working well. There is still a lot of work remaining including halification of existing device abstraction libraries and integration into operating systems and desktop environments.

Notes

1. In general it’s impossible to guarantee UDI stability as the user may connect two identical devices without serial numbers. In this case the UDI’s will still be unique but stability cannot be guaranteed since the user may have e.g. switched the ports the devices are on during power off.
2. In fact this can be implemented by building on top of existing software such as libgphoto or libusb

Putting Accessibility to Work in 2.6 / 2.8

Bill Haneman (with Marc Mulcahy)

This talk presents new accessibility features in GNOME 2.6 and upcoming improvements in 2.8. New developments presented include powerful new scripting features made possible via the combination of python, PyORBit and at-spi.

We'll also show, for the first time at GUADEC, demonstrations of the "complete" desktop via assistive technologies such as onscreen keyboard and screen reader, including Mozilla, OpenOffice, Evolution, and even have a 'sneak preview' of KDE integration.

We'll also take a hacker-focussed look at some accessibility successes and issues from the 2.6 process, and review improvements to accessibility test resources for GNOME developers.

Accessibility and GNOME: How and Why

Accessibility has been a feature of the GNOME 2 desktop since its initial release. Just what that means, and how accessible GNOME really is, has changed with each major release in the 2.0 series. Like internationalization and the Gnome Translation Project (http://developer.gnome.org/projects/gtp), GNOME's accessibility work aims to bring GNOME, and the desktop computing experience, to the widest possible audience.

This is important to GNOME not just because of the potential new base of users who need “accessibility features”; it allows GNOME to be deployed in environments where accessibility to persons with disabilities is already a priority. In particular this includes local and national governmental bodies, where legal requirements to purchase accessible software are increasingly common (see Relevant Laws on the GNOME Accessibility Project site). Without accessibility, GNOME can't be deployed in some environments. It's important to schools, libraries, and community centers that need to serve users with different physical needs and restrictions; software with open distribution licenses is very attractive in these environments, but it must be accessible.

It's important to people with disabilities because it gives users with disabilities more choice; without GNOME, software choices can be quite restricted for people who need accessibility features, as commercial assistive technologies often target only the most commonly used software products in a given class. It's important to existing GNOME users too, since up to one out of six users of software have needs or preferences that can be met by “accessibility” features. When you consider for instance RSI, color-blindness, moderate vision or mobility problems accompanying aging, and temporary disability due to injury, this initially surprising figure becomes easier to understand.

GNOME's accessibility work is also very important for people and assistance programs where money is tight; the cost of proprietary software packages used by a blind user, for instance, can total thousands of Euro or dollars. Differences in pay scales and levels of social assistance can place proprietary accessible computing systems out of reach of people in many countries; and even if cost is not a primary consideration, accessible systems may not be available in the user's native language. In light of the enormous value which computing and information technology can have to users who are blind and users with severely restricted mobility, this can mean an enormous lost opportunity. The GNOME Accessible Desktop represents a choice which potentially removes both economic and language barriers for users who can most benefit from information technology.

GNOME 2 and Accessibility: Highlights

GNOME 2 is special because accessibility support ranging from basic features like theming and keyboard navigation to explicit support for assistive technologies was built into the platform from its initial 2.0 release. This eliminates the need for fragile reverse-engineered approaches. The fact that our accessibility support is API/service-based is expecially important in the GNOME/Linux/Unix environment, where unlike Windows, a very heterogeneous collection of GUI toolkits must interoperate. The client-server basis of the X operating environment also profoundly influences the way that assistive technologies need to work under Linux.

One measure of the success of our architecture is the fact that we now support not only the GTK+/GNOME toolkits where the APIs were first introduced, but also Mozilla, StarOffice/OpenOffice.org, and Java/Swing; KDE has announced that it will support ATK and AT-SPI, and that work is now underway.

GNOME is also special in that it includes full-featured assistive technologies (ATs) in the core desktop packages. This is unusual, if not unique – other operating environments have historically either not bundled ATs at all, or have bundled versions with very limited features. The very existance of full-featured AT as free (FOSS) software is a very significant new development which is receiving a lot of attention.

Current State of Play

The first "evaluation" release of GNOME's bundled AT was released with the GNOME 2.4 series, and continued with the 2.6 release (with the addition of a new AT, "Dasher"). Unfortunately for our many potential users, the AT software in the 2.4 and 2.6.0 releases "could be made to work", but was hardly plug-and-play. Users of early-access distributions of GNOME 2.4/2.6 have reported the "out-of-box experience" for our AT to be mediocre to poor. This may partly have to do with lack of support for packagers of GNOME, or lack of information and experience about how to "smoke test" the accessibility functionality. We have taken some steps to address this issue, and though the core accessibility team still has limited resources for direct support to distros, we think that a 2.6.1-based GNOME can provide usefully functional AT for users who are prepared to run "beta"-grade software. There are also a few issues which are trickier: for instance, many distros will bundle only the Festival text-to-speech (TTS) engine (the only other free option is FreeTTS, which requires Java), and for technical reasons this provides poor quality response for screen reader users. Most screen reader users (accustomed to paying over $1000 for their assistive technology) don't mind paying $25 to $50 for a commercial TTS engine, but there's no place for them to download prebuilt gnome-speech drivers for these engines at this time. Issues like this could use some TLC.

There are other factors which have limited the utility and quality of our accessibility in2.6. Notably, login accessibility has been lacking, though this should be rectified in 2.8. Our theme support to date has been patchy, with few applications apparently being tested by their maintainers against the HighContrast and HighContrastInverse or LargePrint themes. There are issues with applications like xscreensaver (which needs major modifications in order to be used with assistive technologies). There were also a number of scattered regressions, notably as a consequence of changing to gtk+ version 2.4 and the fact that the initial versions of our new widgets were not accessible. This is something which we would very much like to prevent in the future.

To be fair, the ATs until recently were rather more "alpha" than "beta" in quality, so it is unreasonable to point to any one factor which is the blocker. And the story for "ex-GNOME" applications like OpenOffice, Mozilla, and Evolution is still mixed. The good news is that progress is definitely being made in the HEAD branches of all of these products, so the availability of useful applications to accompany the "accessible GNOME" environment is improving.

Prospects for 2.8

So what are the prospects for putting all this to work in the 2.8 timeframe? Is it reasonable to expect that blind and mobility-restricted users will be able to use GNOME 2.8 productively, and to expect that distros and packagers of GNOME 2.8 will provide a working "out of the box" solution?

Here's my assessment:

- recovery from existing regressions looks likely, if module maintainers can spare a little time to review patches quickly and provide occasional API extensions;
- Our A.T.s are likely to reach "1.0" for 2.8, i.e. Self-assess as feature-complete for an initial "product";
- gdm accessibility is on track (but sysadmins and distro packagers will need to work with us to avoid locking out our ATs, allow users to use gdmlogin instead of gdmgreeter, etc.)
- stock xscreensaver will have to be turned off for assistive technology users
  - can be patched to support LoginHelper (see below)
- new X extensions will help, if XORG 1.1 is bundled with 2.8 distros
  - DAMAGE, XFIXES support in gnome-mag
- we are likely to start getting (non-hacker) user feedback and deployments
  - large "accessible desktop" deployments are likely between now and GUADEC 2005
- selected performance hot-spots are likely to emerge, requiring attention

So Is Anything New?

This all sounds like fit-and-polish so far; what's actually new? Well, a lot really. First from the user perspective: GNOME users use more than GNOME. This may seem obvious, but it's important, since so much of the usefulness of GNOME accessibility depends on how it works with office productivity apps, email clients, web browsing, etc.

OpenOffice 1.1 offers prettty good accessibility support (but again, Java is required, which may impact out-of-box accessibility for some distros). The version currently in development will be significantly better, and ought to enable productive use of Writer and Calc by blind and mobility impaired users, and use of the presentation tool via GOK. Mozilla and Epiphany GUIs are now substantially accessible (if you go to HEAD), and the gecko content pane now provides useful, though incomplete, support for content accessibility including HTML forms. Evolution is still lagging a bit, and it's not clear that it will be a good choice for blind users in the 2.8 timeframe, but some blind users are already using Balsa (and coping with a handful of crashers, unfortunately).

KDE support for ATK and AT-SPI is big news, but the existing support only includes the basic accessible widget hierarchy and initial text support – support for AtkAction will take awhile, which limits interoperability with GOK. Unfortunately KDE is still going their own way with services like TTS and magnification – in particular their speech APIs map very poorly onto gnome-speech. We certainly agree that not everyone will want to use CORBA for text-to-speech, but a reasonable mapping between API features in the multitude of speech APIs in development would make better sense. Some service based APIs which we'd like to use or interoperate with present licensing problems – those licensed under GPL present a problem.

As was already mentioned, new Xserver extensions, thanks to efforts by Keith Packard and by Sun engineers, will make a big performance and quality difference as they come on line. Xevie provides low-level interception of keyboard and mouse events without the side-effects of Xgrab, and DAMAGE and XFIXES gives magnification the sorts of notification it needs. COMPOSITE will make an even bigger difference, since it will allow both fullscreen magnification on single-framebuffer systems, and server-side rescaling/post-processing of X pixel data. This will also make possible aids for cognitive disabilities and educational tools – think, for example, of tools that highlight words as they are spoken, for dyslexics, or which can draw attention to on-screen items by “drawing” over client windows. X finally has overlay support (again ;-)).

On the CVS front, there are three new components of note. The first one lives in the at-spi module: the LoginHelper interface already discussed. Also new are two “testbed” projects which, though probably not something that will be integrated into 2.8 GNOME, are likely to lead to improvements in support for blind users. “Gnome-braille” is a project which is developing a general framework for braille encoding, presentation and internationalization; it fills some gaps between the “driver”-like libraries like brltty and libbrl, and fully internationalized braille clients. This is proving to be a very interesting internationalization exercise, especially since many locales don't have “official” braille standards (and many have multiple, competing standards).

One of the greatest challenges to screenreading and usability of heterogeneous applications is the need to application-specific behaviors and customizations. Although we try to provide a uniform user experience, real applications can differ from one another a lot in their characteristics and interfaces, and real users have preferences that vary from application to application. Most commercial screenreaders meet this need via application-specific scripting. Enter “orca”, a module that uses python and C together to provide powerful, customizable scripting of applications and presentation. It's currently both a demonstration of screenreading using Python+AT-SPI and a demonstration of how per-application scripts can leverage the power of at-spi. It also serves as a nice testbed and introduction for newcomers to at-spi, since the Python bindings are considerably simpler than C bindings and simplify much of the reference-counting complexity. James Henstridge presented some Python+AT-SPI work first at LWE in Australia last year, including a demonstration that "spell-checked" the whole desktop – we've extended that idea to an application which actually allows correction of spelling errors (in out-of-process applications) via AccessibleEditableText. It should be clear that all sorts of powerful application scripting can be accomplished via AT-SPI and Python, limited only by the interfaces exposed to AT-SPI. (Currently these interfaces are primarily "GUI" interfaces, but AT-SPI allows for the exposure of non-graphical interfaces as well). Check out ORCA (cvs.gnome.org, module 'orca') and have a look!

Putting GNOME and GAP to work

This talk has been a mixture of "what's hot" and "what's not". What can we as the GNOME community do to put GNOME Accessibility to work?

My top five items:

1) regression control

   we need more testing (by more GNOME hackers!) of accessibility:
   - run with "low vision" themes
   - install, configure, and run your app with the ATs

2) new features, widgets, apps need a11y review:
   - full keynav
   - ATK support
   - theming support + HC icons

3) accessibility should be everybody's job (like i18n)
   - how do we identify barriers to this, and lower them?

2) bundling/packaging and distros: how can the a11y team help support distros?

3) Please read the docs we have, and help us make them better:
   - API docs for ATK and AT-SPI
     - http://developer.gnome.org/doc
     - http://developer.gnome.org/projects/gap/tech-docs
   - Should we create doxygen docs for our IDL?
   - Testing a11y doc from Wipro
     - http://developer.gnome.org/projects/gap/
   - Gnome Accessibility for Developers
     - http://developer.gnome.org/projects/gap/gad
   - Test Assertions
     - We've recently published Sun's "sanity test" assertions for GNOME
     - Incorporating 6 user scenarios

        - http://developer.gnome.org/projects/gap/sanity-testing
        - Use them as a model for adding assertions for <u>your</u> applications

TODO

As always, there are some things left to do...

- some interfaces not yet used:
    - AtkStreamableContent
    - AtkDocument
        - AtkDocument potentially useful for web docs

  these interfaces are mostly for content handling, not UI presentation.

- Languages/l10n:
    - GOK and gnopernicus app strings well localized – thank you!
    - Need non-English voices for gnopernicus
    - braille i18n in progress
    - need to be a squeaky wheel for XKB (to make sure it works well!)
    - GOK needs IIIMF integration

Are We Committed?

I' dlike to close with a question: are we committed to an accessible GNOME? How committed? I can say with some confidence that my company (Sun) is committed, but is GNOME? Sun can' tdo a good job for GNOME by going it alone; if we stand behind our accessibility support, more core hacker involvement is needed. Nothing' s perfect, but we need to grow what we have.

As a corrollary, how much do we care about accessibility users? Are we still a hacker platform at heart? There are hackers with disabilities out there eager to get involved too.. but what about the larger user base?

Interfaces that don' tget used, or which get used only in specialized cases, tend to be undermaintained. Since ATK and AT-SPI are core to GNOME, are there more places where we should be using them? Sometimes this may be the right thing to do even if a new purpose-built interface would have modest technical advantages. Fortunately for all of us, other projects (such as test harnesses, kiosk deployments, etc) are starting to use GNOME Accessibility interfaces, GOK, and other parts of our “accessibility platform”.

BOF
GNOME System Tools

Carlos Garnacho

This talk is a BOF, this means that it will be mostly improvised and dependent on the people participation. Anyway, here's a quick draft of the BOF structure:

The main objective of this BOF is to talk about the future development of the GST, hopefully defining the steps that the tools will take after 1.0, the things that might be worth to talk are:

- Backend/Frontend communication improvements
  Right now the communication is very customized, the frontends read XML from a pseudoterminal, but just before every XML the frontend reads several reports in a custom format about the backend's work, this is mostly avoidable, but there must be a way to make error reporting to the frontends. So the communication should provide:
  - Standard, well documented XML/API
  - Error reporting to the frontends
  - Remote/Local access
  - Easy access from any Language/Platform

  The alternatives proposed (until now) are:
  - Using SOAP
  - Sticking to the current structure, but improving and extending XML
  - Doing communication in both ways

- XML abstraction improvements
  At the moment of writing this document, the GNOME System Tools support the latest versions of RedHat, Mandrake, Fedora, Debian, Gentoo, FreeBSD, Slackware, PLD and OpenNA, so it's not strange to think that several wrong assumptions have been done since the beginning of the tools regarding to distro/OS abstraction. A quick list of to-do things could be:
  - Gather all this experience and create an abstraction layer that truly removes all linux-isms, sysV-ims, etcetera..
  - Get people that really knows about certain distros/OS to do some QA work (mostly for new tools/tasks)
  - get some kind of API stability for letting other tools/modules/whatever trust in the backends
  - document the XML files

- Use of new technologies
  HAL is becoming really nice, and the GNOME System Tools can take advantage of it in lots of places.

- Language replacement in the backends
  Lots of proposals have been done here, right now the backends are written in PERL, this (and maybe the specific system configuration knowledge required) has became a barrier for letting new people collaborate in the GST. so these are are the proposals done:
  - C/Flex/Yacc
  - [C|C++]/pccts
  - Python
  - C#

- New tools
  At the moment there are tools for managing network connections, users and groups, services that run at start time, date & time and bootloaders' configuration, and there is a little team dedicated to a disks tool, but there are still uncovered tasks in the GST. It seems a good time for coding solutions for these, but first we should talk which tasks are :)

- More desktop integration
  While the backends must remain Desktop/Platform/OS/Distro agnostic, the frontends should be more tied to GNOME in order to remove that "GNOME on top of a platform" feeling and eventually get rid of the terminal for system administration

To input many languages

TOKUNAGA Hiroyuki

May 16, 2004

There are many languages in the world, and some of them cannot be inputted from the keyboard directly. Why is this so? The answer is that there are many characters used in such languages. For example, there are over 10,000 "kanji" characters in Japanese and Chinese. Users of such languages use writing systems known as 'input methods'.

Section 1 explains fundamentals of input method. Section 2 explains GTK+'s concrete input method support implementation. Gtk+ has a module system to support input method. Section 3 describes a bit of history of input methods andthe current input method situation in free desktop environment. Section 4 explains uim, a multilingual input method library.

1 Fundamentals of input method

1.1 What is input method?

We define input method as a piece of software to input characters. In many cases, the input method is used to input characters which cannot be inputted from keyboard directly.

Usually, input methods cannot be implemented as standalone software. Input methods require support from applications or GUI toolkits. The reason is that in many cases input method need some infomations held by applications and require applications to forward key events.

1.2 Input method bridge and input method module

Input methods can be split into two logical parts. Because there are no established names for these parts, we will call them "input method bridge" and "input method module" in this paper.

The input method bridge takes care of the communication between applications and input method modules. The input method modules are responsible for receiving key events and convertnig them to corresponding character sequences.

1.3 Class of input method modules

We can classify input method modules by their complexity. Some of them are complicated, but most are not so complicated.

1. Input method modules for some languages (e.g. English) can be implemented by the facility of key-map switching.
2. Modules for some languages (e.g. some of European languages which uses diacritics), can be implemented with a composition table. " + a $\rightarrow$ ä
3. Composition based input method modules have some variants
   (a) Hangul (Korean) has different reduction rule from other compositon based input methods.
   (b) Some input methods for Chinese require candidate selection after inputting compose sequence.
4. Input method for Lao has quite complex FSM (Finate State Machine) due to its complex structure of syllable.
5. Thai input method is distinct. They don't use preedit strings, change surroundings directly.
6. Some of input methods for languages which uses Chinese ideographic characters are extremely complex. (Chinese, Japanese)

The most complicated language for input method is Japanese (and some Chinese input method is complicated, too). For example, a dictionary for Japanese kana to kanji conversion has over 140,000 entries.

1.4 Input method jargon

1. Preedit: The preedit string is a string of characters held by the input method that have yet to be commited as actual input. Applications need to display this string around the current cursor position. When a commit event such as the pressing of the 'enter' key occurs, the contents of the preedit string are transferred to the application to be inserted at the caret position.
2. Candidate: In many input methods, the preedit string needs to be converted to alternate representations. For example, in Japanese, input is accomplished by first typing a word in phonetically, then converting to the correct representation. During this conversion, multiple conversion candidates may match the current preedit string.
3. Surrounding: Strings of characters surrounding the current caret position.

4. Context: Context refers to the client-side state of an input method. There are several ways to manage the context. For example, each GtkTextView has its own GtkIMContext, however an application can exist which shares one GtkIMContext for the entire application.

1.5 Relation between input methods and applications

There are two ways in which the interaction between application and input method can be implemented. The difference between the two are mainly related to the priority in which the iput method receives a key event. We call the two mechanisms “front end” and “back end”.

1.5.1 Front end type

In the front end style, an input method bridge receives a key event before the application. The bridge forwards this event to the input method module, and the module returns values such as preedit strings and commit strings. Nowadays, this type of interaction is rare.

1.5.2 Back end type

In the back end strategy, an application recieves the key event first. The application must forward the key event to the input method bridge, which in turn forwards the event to a input method module.

1.6 The possibility of input methods

Although inpute methods have been used primarily by CJK users, there have the possibility of assisting input for users all over the world.

1.6.1 Prediction

1.6.2 Inline command processing

2 GTK+/GNOME’s input method support implementation

Actually GNOME doesn’t care about input method. So this section explains GTK+’s input method support. GTK+ has a class GtkIMContext and a module system called immodule. In our wording, GtkIMContext corresponds to the input method bridge, and immodule to the input method module.

An input method module is a shared library implemented as a child class of GtkIMContext. This is called an immodule. There are some immodules in the GTK+ official release tarball, and many modules that are not included with GTK+.

The initializing function for the immodule system reads the file /etc/gtk-2.0/gtk.immmodules (This file location can be overwritten via an environment variable `GTK_IM_MODULE_FILE`) and loads immodules indicated in this file.

Text widget calls the function `gtk_im_multicontext_new` to create an instance of GtkIMMulticontext, which is a subclass of GTKIMContext.

2.1 From the point of view of the application (widget) developer

First, if you don't write a new widget from scratch which deals with texts, you need not care about input method.

GTK+ has an abstract base class GtkIMContext. And there are two classes, GtkIMContextSimple and GtkIMMulticontext which are a child classes of GtkIMContext. The creator of a text input widget needs only to worry about the following steps:

1. Create an instance of GtkIMMulticontext by calling `gtk_im_multicontext_new`.
2. Write proper callback functions to correspond some signals. Unfortunately, there is no reference document about how to do this completely.
3. If a key event is received, send the key event to IMContext first by using `gtk_im_context_filter_keypress`. If returned value of the function is FALSE, then process the key event.

3 Present input method situation in the free desktop environment

3.1 A bit of history

To understand the situation related to input methods (here it means especially input method bridges), we should trace the history at first.

June 1994, XIM (X Input Method) was introduced at X11R5. XIM had some problems and needed some improvements, but XIM the development team was isbanded by the Opengroup's absorption the X Consortium. Development of XIM was stopped, but an alternative did not appear for several years, and many people have resorted to using XIM even now. In Japan, some people are calling the time between the end of XIM development and 2000 the "input method dark ages."

Septempber 2000, IIIMF (Internet/Intranet Input Method Framework) was released from Sun Microsystems. IIIMF had resolved some problems of XIM such as removing the strong dependency to X, but had some other problems.

November 2000, the immodule system was introduced to GTK+ 1.3. By doing so, legacy IM code were removed from libgtk and GTK+ was ready to support new input methods.

Aug 2002, uim is released by Yusuke Tabata. Uim aims to resolve the problems XIM and IIIMF had. It's implemented as a shared library and doesn't use TCP/IP.

3.2 Current situation

The input method situation of GNOME/GTK+ was explained in section 2. Here, we will describe the situation for KDE and OpenOffice.org.

KDE is very famous desktop environment and there's no need to introduce it. Because KDE uses Qt as a GUI toolkit, the important thing here is the input method support of Qt. Currently, the latest version of Qt is 3.2.3, and it supports only XIM. Moreover, the support is a bit buggy. Daisuke Kameda has made a patch for Qt (5) to implement a module system similar to GTK+'s immodules. It's unclear that whether this patch will be accepted for Qt3. However, since developers of Qt seem recognizing necessity of new input method support, some form of the patch will go into Qt4.

On the other hand, now OpenOffice.org supports only XIM. Because OOo is a cross platform software, it must have an abstracted input method support, but unfortunately it seems there is no implementation for other input method library/framework supports.

The big two GUI toolkits have implemented a modular system for input methods. So, we can now say that the input method bridge has been separated from applications.

We are currently in a transition stage. In the next several years, de facto standard of input method bridge will be decided.

4 Overview of uim

Uim is a lightweight, but fully featured multilingual input method library. It is implemented in C, and has a built-in interpreter for a subset of the scheme language. Goal of uim is to provide a secure and useful input method libarry for all languages in the world.

Uim can be split into three parts: a core library, bridges, modules.

Uim currently has a few bridges and works in many environments. There are bridges for GTK+ (2.x), for XIM, for Qt (this need the patch 'immodule for Qt'), for GNU Screen(screen-uim), for console (uim-fep), for Sharp Zaurus (IMKit-uim), and for MacOS X (MacUIM).

4.1 Policies of uim

4.1.1 Uim does not use TCP/IP

Uim does not use TCP/IP, this is important thing from the point of view of security. Because an input method recieves all key events, so using TCP/IP for an input method introduces the risks of keystrokes being sniffed through a security hole. Once an input method is cracked, you may even have you SSH key passphrase stolen by such a flaw.

4.1.2 Simple library

Inputting of most languages is not so complex, and there is little necessity of taking a client-server system for such languages. So uim is a shared library, not a client-server system. Client-server system should be implemented on the input method module layer, if need.

References

[1] Uim official web page
http://uim.freedesktop.org/

[2] IIIMF official web page
http://www.openi18n.org/subgroups/im/IIIMF/

[3] Programming for Japanese characters input
http://home.catv.ne.jp/pp/ginoue/im/index-e.html

[4] Pango input resources
http://www.pango.org/input-resources.shtml

[5] Immodule for Qt
`http://www.kde.gr.jp/~daisuke/immodule_for_qt/pukiwiki/?ImmoduleForQtDocsForTrolltechPass`

GNOME in Japan

Yukihiro Nakai
Japan GNOME Users Group

1. History

GNOME is a popular desktop also in Japan. Japan GNOME Users Group (JGUG) is the first GNOME users group in the world. I founded JGUG in 1998, with the background of the initial official FreeBSD GNOME binary maintainer and the first Japanese GNOME messages (ja.po) translator. FreeBSD has supported GNOME officially since GNOME 0.30 with Japanese translated, rather earlier than Red Hat did with Red Hat Linux 6.0. (Red Hat Japanese support started from 6.1, with very broken quality. I joined Red Hat from Red Hat Linux 7.0 Japanese Edition.)

At that time, GNOME was already major than KDE in Japan; Troll Tech's Qt lisence was so restricted to prohibit to release i18n patches for it. Japanese People were waiting the completely open and free desktop and toolkits. Chinese and Korean ignored Qt's lisence and distributed the illegal patches. So KDE is still major there, Chinese and Korean GNOME community site did not happen until very recently.

2. Activities

We have 3 domains:
  gnome.gr.jp
  gnome.jp
  ja.gnome.org

Currently we use gnome.gr.jp and have 3 public mailing lists:
  gnome-users@gnome.gr.jp
  gnome-devel@gnome.gr.jp
  gnome-translation@gnome.gr.jp

gnome-tranlation has highest activities among this. Around 5 people has cvs.gnome.org accounts. 99.99% messages(9th) are translated for GNOME 2.6 and 78.18%(16th) for GNOME 2.8. The rest untranslated for GNOME 2.6 are unstraslatable ones in Gnopernicus, which don't consider word order difference between English and Japanese. ('%d of %s' or something like that)

3. Future roadmap for GNOME and Japanese

3.1 Fix Japanese failures in GNOME

Japanese are traditinaly known as very sensitive users among UNIX communities. Be upset if messages are not translated into Japanese, get pale if it doesn't handle Japanese well, feel the end of the world if the latest release isn't shipped with enough fixes for Japanese issues. Some GNOME application developers have met such incomprehensible complaints from Japanese users.

- Gpdf CJK

  Gpdf is GNOME port of xpdf and bonobo enabled. Xpdf support CJK pdf (bitmap font only) but gpdf deleted all CJK features. For wronger,

GNOME set this as the default pdf viewer. Ghostscript and ggv support
CJK but bonobo-nized pdf viewer is still required.
Pdf uses CID for its string encoding, freetype cannot handle CID font
but only supports Unicode encoded TrueType font. So it is need to add
CID and TrueType font mapping.

- Evolution

  Evolution is known as nothing in Japan because it doesn't support
  Japanese well. If mail has circle digit characters, which are popular,
  or other JISX0213 chars, it crashes. JISX0213 handling in mails is defined in
  iso-2022-jp-4, and now updated to iso-2022-jp-2004. GNU libc also needs
  to support iso-2022-jp-2004, but Evolution should not crash on other
  iconv platforms anyway.

- Mono

  Mono implements C# classes of Japanese features according to Microsoft' s
  specification document faithfully, but it crashes when it get Japanese
  calendar strings as its input. Japanese calendar means Japanese emperor' s
  era, not localised strings of Gregorian calendar.

- Anjuta

  Anjuta uses Scintilla widget as its editor compoment, but it' s broken
  for i18n.

- Pango Japanese Module

  Pango default language module has minimal Japanese hyphenation support,
  but not enough. And to support the furigana feature, which shows
  small phonetic characters beyond strings, Japanese specific pango module
  should happen.

- Gnome-Print PDF CJK

  Gnomeprint support CJK Postscript but not for PDF generation. A nice
  Sun guy announced to implement it, so just wait it happens.

- Gnumeric

  Gnumeric is a GNOME spread sheet and full-featured GNOME office tool.
  It aims to implement all Excel features so it lacks many Japanese
  features. Furigana support, Japanese calendar support and Japanese Excel
  functions are expected.

- Gedit

  Gedit is the text editor. Emacs is doing a lot of things as an editor
  for CJK handling, so gedit should do. Codeset autodetection, codeset
  conversion are the examples.

- Gnome-terminal

  Gnome terminal supports CJK well but a failure for Japanese ideograph support is found recently.

There are other Japanese issues, of course. Some GNOME tools handle Japanese better than other open source software, but most are much worse than the corresponding Windows free or commercial software. Additionaly, Japanese users complain about Japanese feature without looking what is really essential point. Red Hat Linux 8.0 introduced Anti-Aliasing with GTK+2 and Japanese market made a big reaction. People said they felt Linux got easy to install than ever before, but actually the change was just Japanese True Type font instead of bitmap font, and GNOME Japanese fault are still there.

3.2 GNOME Website Localisation

Translation work do not finish within software. Website is the important part of GNOME, so http://www.gnome.org/ localized official website is neccessary.

Footnotes (http://www.gnomedesktop.org/) is anothor indispensable GNOME website. There is localized site at (http://footnotes.gnome.gr.jp/). But Footnotes uses phpnukes, which i18n feature is buggy and not maintained. I suggested to use other community site web package for i18n support, but the maitainer doesn't respond.

3.3 GNOME Foundation Japan

GNOME Foundation Europe looks like failed to settle because most activities are redundant. Companies do not want to donate duplicated foundation.

My idea for GNOME Foundation Japan is different. Just focus to raising travel fee to Japan for GNOME speech. There is no inconsistency with the activity of GNOME Foundation. Companies or groups, who want to invite GNOME hackers to Japan, pay 1,000 USD each, and 5,000 USD might be enough to spend 1 week in Japan with his/her family. Companies will easily understand this raise do not take effect to get services by GNOME Foundation.

3.4 GUADEC in Japan

GUADEC is a popular GNOME Conference in the world, and what I want to invite to Japan. It might be difficult because GUADEC's E is Europe, and people do not want long time flight to Japan. And travel fee is rather higher.

Local GNOME Conference is rather actual and reasonable but GNOME hack activity is still very low in Japan and neighbor countries. Japanese govenment is tring to increase open source hackers in Japan now, but it will not happen so soon. So, start from small session or 1 track in other conference might make sense.

4. IPA Open Source Project

Information-technology Promotion Agency(IPA), an incorporated administrative agency owned by Japanese government, started open source project. They call project themes from public, and make programmers develop the software with their budget. It started at 2003 fiscal year and continues 3 years initially. The rumor is saying the budget total is 1 billion yen (around 9 million US$)

and the purpose is to replace all desktops in Natinal Institute of Advanced Information Science and Technology(AIST), another
governmental agency.

As one of the project theme, I implemented Advanced Japanese features to open source office suite. The target was not OpenOffice.org, but gnumeric.
Implemented features are below:
- Furigana support
- Japanese hyphenation
- Japanese spellcheck
- Japanese syntax check
- Japanese calendar support

GNOME and Usability in Military Systems

Kathleen Fernandes, Ph.D.
Space and Naval Warfare Systems Center, San Diego

Introduction

In February 2003, the U.S. Department of Defense (DoD) certified a Red Hat Linux Common Operating Environment kernel, thereby passing a key milestone in allowing the use of Linux in command, control, communications, computer, and intelligence (C4I) systems. To achieve certification, the platform had to present a specified appearance and behavior (i.e., "look and feel" as well as functionality). While the certification may be viewed as evidence of the enterprise readiness of this open source software, more work remains for GNOME to achieve a level of desktop maturity comparable to that of other industry offerings.

This paper focuses on the maturity of GNOME as it relates to the needs of the warfighter who must perform decision making in a range of operational settings, including ones that are sometimes hostile. A C4I system provides the capabilities required by military commanders to plan, direct, and control the operations of forces to execute their assigned missions. A C4I system fuses inputs received from various external sensor, intelligence, and environmental sources into a near-real-time operational picture that can be used to support mission-related tasks such as time-critical targeting, missile defense, fire support, mission planning, communication management, and war gaming and simulation.

The C4I User Environment

C4I systems in the DoD are designed for use on multiple hardware platforms with varying configurations of display and input devices. Users have access to a mix of commercial and military applications developed using various graphical user interface toolkits. Users may interact with a C4I system on client workstations in a command center with an office-like environment, or a system may be deployed in a tank or onboard a ship or submarine or installed in a mobile or field-based shelter for use by forward deployed units. Workstation hardware may be "ruggedized" for survivability, and users may have to interact with a system while wearing protective gear. C4I systems place significant demands on users for speed and accuracy in performing mission-related tasks while functioning in a high-stress decision environment.

A C4I system provides a unique microcosm within which to examine and assess GNOME capabilities. By focusing on a domain where users operate in a sometimes hostile environment and where the penalty for errors can be severe, the open source community can better understand the impact of usability shortfalls on operator performance. With regard to the need for systematic user interface and user testing, Eric Raymond indicated in "Why Open Source Will Rule" that "we're not very good at that yet, we need to find a way to be good at it." Steven Pemberton made a similar statement in "Interactions" magazine, indicating that the open source community "can't do usability" because the software is produced by programmers for programmers and does not meet the needs of the general public.

Usability in C4I Systems

The usability issues described in this paper are drawn from a comparison of GNOME, Java, Motif, and MS Windows styles that was completed in December 2003 and documented in user interface design guidelines for C4I applications. Rather than looking at how users perform tasks using the capabilities provided by GNOME, this comparison focused instead on the usability of individual interface components. The following usability issues were identified as a result of this comparison:

(1) GNOME lacks detailed documentation of its "look and feel" that would allow an interface designer to produce a "HIG-compliant" implementation. While the GNOME Human Interface Guidelines contain generic design guidance for application developers, it does not provide the level of detail normally covered in such a document. In particular, the HIG does not document the behavior attributes of individual interface components. As a result, when interface designers examine the style provided by a GNOME implementation such as Red Hat, the absence of detailed documentation makes it difficult to determine what is a feature, a bug, or a variation in implementation.

When the basic "look and feel" attributes of an interface are not defined, interface designers cannot determine if their implementation will provide a style that is consistent with other desktop applications available to users. In addition, while interface designers can adopt a “discovery” approach to derive their own description of “look and feel” attributes, this process can be time-consuming and must be repeated each time an update to GNOME is released. The GNOME community needs to both document the attributes of individual interface components as well as provide design guidance that explains how to use these components to create an effective interface. While the GNOME community may resist the constraints imposed by these requirements, interface designers outside this community would embrace the guidance and incorporate it as a key element of their interface design process.

(2) GNOME does not prevent users from making errors in performing basic actions such as navigation and selection. The style comparison found that while GNOME interface components provide consistency in behavior for documented actions, legacy behaviors that are inconsistent with this model are also available to users. For example, while the left mouse button selects objects, the other mouse buttons also perform this action. In addition, GNOME does not provide a consistent mapping of actions to mouse buttons or individual keys on the keyboard across interface components. For example, clicking on a list item using the middle or right mouse button selects the item, but clicking on a push button, radio button, or check box using one of these mouse buttons moves focus to the

control. In addition, both the Space and Return keys can perform a select action depending on the interface component with focus.

In the absence of detailed “look and feel” documentation, it is difficult to determine if the inconsistencies in behavior are built into the toolkit or specific to the implementations examined. Most users can easily recover from the consequences of inadvertent or random actions. However, for military users, the tasks they perform and the operational environment in which they work increase the likelihood that these types of errors will occur and their consequences will be severe. The GNOME community needs to attend to and correct these interface behaviors, given their potential impact on the user performance.

(3) GNOME relies on themes to impart a common visual style to its desktop that contains applications created using a mix of toolkits. Joe Spolsky indicated in “User Interface Design for Programmers” that the capability of themes to change the appearance of an interface without changing how it behaves is appealing because “users are completely free to ignore the choice and get their work done anyway.” The flexibility provided by themes presumes that the toolkits used to create the applications available to users support similar behaviors. The style comparison indicated, however, that while the toolkits examined support many common “look and feel” attributes, each also defines unique rules for the appearance, behavior, and layout of interface components.

A theme affects the appearance of a user interface but not its behavior or layout. A hybrid style is created when an interface component has the appearance of one toolkit but the behavior and/or layout of another. While themes increase the visual uniformity of a user interface, they can create a hybrid style by removing the visual cues that indicate to users where there may be differences in behavior and/or layout. For example, the style comparison indicated that when users interact with a GNOME desktop, they may encounter windows that are similar in appearance but differ in selection methods, function key assignments, and push button order.

The lack of user interface predictability can reduce the speed and accuracy of user performance and impact the development of expertise by users. As indicated previously, while the generic user can accommodate to style inconsistencies caused by these differences in layout and behavior, the consequences for a military user can be significant, given the unique demands placed on them for speed and accuracy in the performance of mission-related tasks.

Conclusion

GNOME is making the transition from desktop software designed by and for the open source community to a collection of desktop capabilities that has seen significantly wider adoption by individuals and institutions worldwide. The intent of this paper is to motivate the continued evolution of GNOME by focusing developer attention on the requirements of a user community that must be able to perform effectively in a sometimes unfriendly operational environment. By doing so, it is hoped that GNOME can achieve a level of desktop maturity that will make it a viable option for other communities with similar user requirements.

References

Broersma, M. “Eric Raymond: Why Open Source Will Rule,” March 2002. http://zdnet.com.com/2100-1104-871366.html

Defense Information Systems Agency. “Common Operating Environment (COE) User Interface Specifications (UIS), Version 4.3,” December 2003. Document available by request.

The Mitre Corporation. “Use of Free and Open-Source Software (FOSS) in the U.S. Department of Defense, Version 1.2.04,” January 2003.

Pemberten, S. “Scratching Someone Else's Itch (Why Open Source Can't Do Usability).” Interactions, Jan/Feb 2004, 72.

Red Hat. “Red Hat Achieves Defense Department COE Certification,” February 2003. http://www.redhat.com/about/presscenter/2003/press_coe.html

Spolsky, J. “User Interface Design for Programmers,” October 2001. http://www.joelonsoftware.com/navLinks/fog0000000247.html

GNOME Community Road Map

Introduction

GNOME is a large Open Source software project, with hundreds of contributors from right across the globe. Some of these contributors work on GNOME as part of their employment, but the majority offer up many hours of their free time and energy in helping to create one of the premier desktop environments for Linux and UNIX-like installations.

The GNOME Community Road Map is a big-picture view of what functionality GNOME can expect to include through the next year and beyond. The Road Map is a combination of feedback from current GNOME developers and other community members.

Due to the largely volunteer nature of the project, constructing a detailed, long-term list of concrete goals is impractical. However, it is possible to discuss the general themes that drive development of GNOME.

For specific pieces of GNOME, there are a number of concrete short- to medium-term goals listed. This list of improvements comes from the individual module maintainers and contributors.

This document describes the major themes of GNOME development. Each section describes theme and why it is important. It then lists three sets of concrete tasks - 2.6 Improvements, 2.8 Plans, and Long Term Goals. Items listed in the 2.6 Improvements sections are already implemented and will be available in the GNOME 2.6 release. Items in the 2.8 Plans are tasks that a contributor has expressed interest in implementing, and can be expected for the 2.8 release. Long Term Goals list items that the maintainers have identified as important, but for which there are no concrete plans to address in 2.8.

Universal Access

The GNOME desktop strives to be a productive working environment for all people. Wherever possible, GNOME strives to eliminate obstacles that unnecessarily hamper the user experience. It is a guiding principle of GNOME development that software should operate smoothly, regardless of the user's level of expertise, language preference, or physical disabilities.

2.6 Improvements

* New file selector from GTK+ 2.4.

* Bundled assistive technologies are more functional, and new assistive technology and features have been added.

* Existing accessibility support has been largely or completely internationalized.

* Integrated support for internationalized keyboards in control-center.

* Improved user-visible mime UI.

* Object-oriented ("spatial") file manager to improve usability.

2.8 Plans

* More improvement to the user-visible mime U.

* Accessibility support interoperating seamlessly with Web, email, office applications, and other GUI toolkits.

Open Questions

* Mozilla ships the Firefox browser, which has similar goals to GNOME's Epiphany browser. We would like to work with the Mozilla foundation to settle on a common direction for the web browser.

Collaboration

Communicating and working with other people is not simply a function of a single application that sits in a rectangular window on your screen -- Evolution or Outlook, for example -- but one of the primary functions of a computer. Therefore, collaboration should be a first-class element of the user experience.

The GNOME desktop will have collaborative elements woven throughout it: a centralized presence icon mechanism that shows whether people you are working with are online or not, the ability to share files and data with people from anywhere in the desktop, and generally the presentation of collaboration-related information in every part of the desktop where it is relevant, not just in one or two applications.

2.6 Improvements

* Optional integration of contact and calendaring information into the desktop.

2.8 Plans

* Inclusion of Evolution into the GNOME Desktop.
  - Inclusion of the Evolution data server will provide a central touchpoint for addressbook and calendaring information.
  - The Evolution client will provide mail, addressbook, and calendaring capabilities to the desktop.

* Wide-spread integration of addressbook and calendaring integration with the desktop.
  - Integration of GAIM contact list with addressbook

* Easy sending of files over email and IM.

* Discovery of network services.
  - Detection of file shares through rendezvous will be integrated into the gnome-vfs volume-management layer.

* Integration of presence information into the desktop.
  - There are two projects underway (gossip and galago) to provide presence information integrated with the addressbook.

* Personal file searching based on local file index, including file metadata searching based on the Medusa file indexer.

Long Term

* Blogging integration.

* Peer-to-peer data sharing.
  - It should be easy to create shared workspaces between team members without intervention from a system administrator. Novell' s iFolder is one possible implementation.

* Metadata framework

- Possible implementations include Novell's Simias, GNOME Storage

Media

There are still pending issues with the stability, legality, licensing, and/or developer-friendliness of the available the media framework solutions. Partially due to these issues, no media applications have been formally included in the GNOME 2.6 release. However, the community is actively developing a complete series of media applications.

2.6 Improvements

* Sound-Juicer, a CD audio importing tool has been developed.

* The Rhythmbox music player has matured and is now more stable and feature complete.

2.8 Plans

* Integration of CD audio importing into rhythmbox.

* Integration of CD burning into rhythmbox.

* Integration of portable music player support into rhythmbox.

* Investigation of legal issues surrounding licensing of media frameworks.

- There is an explicit goal that the platform be LGPL or more liberal in order to allow ISVs to use the platform to create proprietary software. This policy has never been formally documented. Both of the proposed media options have open legal questions. The Helix framework is not LGPL- or GPL-compatible. It is unclear that the Gstreamer license allows for proprietary codec implementations.

Long Term

* Movement of media widgets into the core platform.

* Audio server replacement.

* Better image viewing/photo collection manipulation.

Hardware

One area in which GNOME has lagged behind other desktop operating systems like Windows and Mac OS X is tight integration with hardware. GNOME is working with the freedesktop.org community to make plug-and-play hardware management just work.

In addition, the GNOME community is working on supporting a wide range of devices - for example digital cameras, music players, and bluetooth devices.

2.6 Improvements

* Better handling of removable media in GNOME VFS and the file manager.

* DVD burning support in the Nautilus CD burner.

2.8 Improvements

* Better hardware integration via freedesktop.org's D-BUS and hardware abstraction layer (HAL).

* CUPS management tools.
  - Ximian has developed a set of tools to manage CUPS printers and integrate them into the printing subsystem. These should be integrated into GNOME for the 2.8 release.

Manageability

Manageability of the desktop environment is a key area for most large-scale deployments. The ability to centrally and remotely administer settings of core desktop applications, along with the ability to "lock down" end users' desktop configuration, are important parts of an enterprise desktop.

GNOME is working toward providing these capabilities, both in the underlying platform and in specific applications.

2.6 Improvements

* Lockdown mode for panel configuration

* Web Browser lockdown mode

* Improved lockdown and management via GConf

2.8 Plans

* Improved menu system, including better compliance with freedesktop.org specifications.
  - Allows easier management of panel menus

Long Term

* Improved gconf API and partial daemon rewrite.

Core Platform Improvements

GNOME realizes that ISV adoption is necessary to the success of the free desktop. As such, constantly improving the development platform while keeping it API and ABI compatible is an important goal.

An important part of improving the free software development platform is communication and cooperation with other desktop environments.GNOME is heavily involved with the freedesktop.org initiative. Within the freedesktop.org project, GNOME works to develop standards and implementations that ISVs can develop against to build applications that work well regardless of the user' s desktop environment.

2.6 Improvements

* Introduction and adoption of the GTK+ 2.4 toolkit, including the new file selector API and UI, new and easier menu API, and some other new widgets.

* gnome-vfs daemon, improved authentication and connection sharing.

* Improved mime-type detection and database (as part of freedesktop.org).

2.8 Plans

* Database support.
  - The gnome-db project is focusing on adding database support to GNOME office applications.

* Inclusion of D-BUS, a system-wide messaging daemon.

* Improved accessibility documentation, which will help all developers ensure that their applications are completely accessible.

* Release of GNOME Human Interface Guidelines version 1.2.

* Clarification/improvement of Nautilus extensibility APIs.
  - A new extension API was added to nautilus in 2.6. They will more widely adopted and tested in the 2.8 timeframe. This extension API is not currently a part of the core platform.

* Formalization of the Platform licensing policy.

Long Term

* Deprecation of libgnomecanvas and libart in favor of new API based on the Cairo library

* Improved cut and paste/drag and drop format documentation, to allow better interoperability in this area.

* Improved applet/tray icon API and usage guidelines.

* New VFS API, to allow for less UNIX-like file semantics which are easier to develop against and more appropriate for a larger class of VFS backends.

Open Questions

* GNOME is currently implemented in C, with language bindings implemented for use in third-party applications. There is some consensus in the community that adoption of a higher-level language and runtime would be beneficial to the development of the desktop.

Java and C# have been proposed as alternatives. The community is currently discussing the technical, political, and legal ramifications of adopting these languages into the desktop.