Octave-GTK: A GTK binding for GNU Octave

*Muthiah Annamalai, Hemant Kumar,C Ramasamy,R Saravana Manickam,Leela Velusamy**
National Institute of Technology - Tiruchirapalli, India
email: {ec10130,cl10112,cl10120,me10143,leela} @nitt.edu

25^{th} November 2004

Abstract: *Language binding, is a favorite solution for programming language interoperability problems, that helps extend the reuse of libraries, and save developer's time, all the same providing new functionality to the host language . In this paper, we discuss the problems faced with interoperability between two programming languages, with respect to GNU Octave, and GTK API written in C, to provide the GTK API on Octave as a part of Octave-GTK, our research project. Octave-GTK is the fusion of two different API's exported by GNU Octave [scientific computing tool] and GTK [GUI toolkit], to use GTK primitives within GNU Octave, to build graphical front ends,at the same time using octave engine for number crunching power. Octave GTK, is a GTK binding for Octave. The concept of binding is prevalent in the world of free software, and this paper illustrates our implementation this technologies, and binding logic. Also shown, are methods of code generation, binding automation, and the niche we plan to fill in the absence of GUI in Octave. Canonical discussion of advantages, feasibility and problems faced in the process are elucidated.* [1]

General Terms: *GNU,Language Interoperability, Library reuse, Software Engineering*

Keywords: *Free Software,Language binding, Interoperability,Octave-GTK, Code Generator*

1 Aims

Octave GTK+ is a project that aims to add GTK+ bindings to Octave by extending it, and build a GUI for Octave around these features of GTK+. Our aims are two fold.

1. Generating the GTK binding
2. Working on a Octave GUI

2 Overview of technology

The target API, GTK is written in C, whereas the host language GNU [5] Octave, is interpreted. This complicates things further, as we will see [sec 4]. We show all our results on a GNU/Linux system running Linux kernel 2.4, on a x86 processor.

To understand the relevance of Octave-GTK, and the problem it solves, it is essential that the reader be familiar with GNU Octave, and GTK technologies, and the space that Octave-GTK is to fill.

2.1 GNU Octave

G̈NU Octave is *high-level language, primarily intended for numerical computations. It provides a convenient command line interface for solving linear and nonlinear problems numerically*ïs how GNU Octave[2] describes itself.

GNU Octave is a large project,with 146,875 lines of C++ code, few FORTRAN programs borrowed from standard numeric libraries like *lapack,blas,fftlib,ranlib*, and*ODESSA*, a lexical analyzer and parser written in flex and Bison, an Octave interpreter for the Octave language.

GNU Octave also supports an interpreted language called octave, which has access to the whole lot of octave libraries for number crunching. In fact much of the octaves functionality itself is written in octave.

GNU Octave has capacity to solve Ordianry Differential Equations(ODE), perform symbolic computing, plot 2D and 3D graphs, and its basically a high level computational tool for the scientist and engineer. Also, special packages for Image, Signal, Audio processing exist in Octave Forge[4].

Octave accommodates extensions, by using shared libraries, *dynamic loaded files* that provide extra functionality. This is similar to the *plug in* concept.

[1] [*] corresponding author [leela@nitt.edu]
Department of Computer Science and Engineering, National Institute of Technology, Tiruchirapalli

Octave can take a shared library and load all the symbols(functions and variables) present in it, to extend its[Octave's] functionality to the user. Thus GNU Octave package gives us the power to extend Octave interpreter, and utilize the inbuilt computational routines, for other needs; say like GUI for scientific programs, where we cannot expect every GUI programmer to write his/her own Matrix routines, Plotting functions et-al. This is where Octave-GTK hopes to play the vital role, by bridging this gap.

2.2 GTK

GTK [Gimp Toolkit] is a cross platform object oriented (*OO*) GUI toolkit, written in C, as a collection of several libraries *glib* , *gdk*,*gdk-pixbuf*, and *gtk* itself, altogether nothing less than, 358,998 lines of code.

One of the design goals for writing GTK in C, was that it would be easy for others to write language bindings for scripting languages. Given the fact that many scripting languages themselves are implemented in C this is considered feasible, if not easy.

2.3 Other GTK Bindings

Proof of the design is, seen in the number of language bindings for GTK, from the languages like *lisp, guile, Ada,SLang, C++, C, Python, and Perl.*

2.4 Octave-GTK

One see that Octave-GTK is a technically feasible problem, trying to export the GTK API to be acessible from the GNU Octave runtime. As with the like of GTK bindings, [sec 2.3] one can be convinced that GTK binding code, can be written from the compiled languages [C++], interpreted languages [list], or both [Python]. Octave-GTK design and architecture will be presented in the following sections.

3 Problem definition

Definition: To create a language binding for GTK from Octave, to access GTK function from Octave language and interpreter.The steps involved are

1. Translate Octave types to C, for access from within GTK API.
2. Translate GTK C objects into Octave Objects for access from Octave.
3. Make GTK API functions, accessible/callable from octave language
4. Make Octave functions,both builtin & custom, be callable from C, for use as callbacks.

4 Architecture

The specifications derived in [sec 3], dictate the architecture used to solve the problem. We have to make a *glue layer* implemented for Octave interpreter as the *octave-gtk* shared library. As shown in the [figure 7], this *octave-gtk* library does all the work mentioned in [sec 3]. This is our *mechanism.*

5 Prototype

We have made a *proof -of-concept* implementation of the *octave-gtk* glue layer proposed. The image [figure 8], shows our first implementation. The section of Octave code given below, produces the output in this [figure 8].

```
#! /usr/bin/octave -q


function click_cb(widget)
     w1=gtk_window_new("I Love GNU");
     b1=gtk_button_new("Me Too");
     gtk_container_add(w1,b1);
     gtk_widget_show_all(w1);
     g_signal_connect (w1,"destroy",
                       "gtkwidgetdestroy");
     g_signal_connect (w1,"enter_notify_event",
 "gtkwidgetdestroy");
     g_signal_connect (w1,"key_press_event",
 "gtkwidgetdestroy");
end

function main()
     printf("Welcome to Octave-GTK+\n");
     gtk_init();
     w=gtk_window_new("Octave-GTK team.");
     b=gtk_button_new("So What?");
     gtk_container_add(w,b);
     gtk_widget_show_all(w);
     g_signal_connect(b,"clicked","click_cb");
     g_signal_connect(w,"destroy","gtkmainquit");
     gtk_main();
end

main();

%octave-gtk demonstration.
```

We have loaded the glue code all at once by calling *gtk_init()*, from the Octave runtime. This loads Octave interpreter's symbol table with all our glue functions. Within the glue functions themseleves we have a pattern of calling the target GTK functions.

5.1 Glue logic

1. Check, of arguments are valid
   i.e dont take integer when object expected, et al
2. Translate Arguments
   Convert Objects to pointers, string to *char **
3. Call the GTK function
4. Return Arguments to Octave
   Inverse of step 2. Convert pointers to Octave Objects

```
/*
C Prototype
GtkWidget *gtk_button_new(const char *name);
*/
DEFUN_DLD(gtk_button_new,args,,
"creates a Button")
{
  string ss;
  long int x;
  GtkWidget *w;

  if(args.length() < 1)
    {
      std::cout<<"eg: gtkbuttonnew
   (title)"<<std::endl;
      return octave_value(1);
    }


  ss=args(0).string_value();
  w=gtk_button_new_with_label
   (ss.c_str());
  x=(long int )w;
  cout<<"Button created Created"<<endl;
  return octave_value((double)x);
}
```

First of all the we get the char name for the button from the octave interpreter and with in the glue code,we check if the data supplied by the interpreter is char string or not.After that is done we go to the next level and create the button using the C function.After that the pointer to the button is returned to the interpreter as octave_value.

5.2 Code Generation

Since all this process [sec 5.1] of glue logic is same for all the functions called, we may have generalization of the concept, and introduce code generators to do the job of producing the glue code. This is nothing new, as seen from various GTK bindings mentioned in . For most functions this will work, when we give a *type-mapping* between Octave types and corresponding C types.

Type-mapping is helpful for representation of GTK objects [sec 2.1], using custom designed Octave objects, by deriving from types like *octave_base_scalar*. Thus, we can store pointers within a member of the Octave object,so also its GTK attributes likes widget name, type, properties can be stored. Now *type-mapping* should be considered solved.

Concept of *type-mapping* is not always applicable for all C constructs, where pointers to integers, might mean returning a single variable, array or some typecast value. This, cannot be understood by the code-generator, even with *type-mapping*; so in some cases we have to provide manual overrides for the rest of the functions, that have ambiguous *type-mapping*.

Ideal choices for code generation for the GTK API as shown in [6] binding is *Python* language. For feasibility of codegeneration for Octave, we experimentally target the GD[8] library and produced the [9] GD-Octave glue for Octave.

5.3 Implementation

In this prototype, we have not used custom octave objects, no typemapping. This means, simply, **we store the GTK widgets, and pointers, in long integers**. Naturally this technique is *non-portable* and user can easily crash the Octave interpreter, by passing wrong pointer (often the cause of segmentation faults). There has been no type checking but the other steps 3,4 of the [sec 5.1] have been followed. As with the case of evolutionary software, we are in the process of implementing the refined designs, detailed earlier.

5.4 Callbacks

This presents one of the biggest challenges in the problem. Calling an Octave function, that is a callback from the GTK environment. Every widget, which needs a callback, stores within its member variables, the name of the Octave function acting as a callback for a particular event. An intermediate generic callback is registered with the GTK system. This intermediate function which is the callback from

the GTK side, extracts the name of Octave callback function from the widget (we store it earlier), and then uses Octave interpreter to evaluate that function using *feval()* .

Our method of implementing callbacks are heavily dependent on the introspection capabilities offered by GNU Octave. We will be using the Octave interpreter's symbol tables, and function lists to findout the callback and functions like *feval()* to evaluate the function callbacks with necessary arguments.

6 Advantages

When the Octave-GTK code is complete, advantages of this language Interoperability with Octave & C are as follows.

1. Octave will have a GUI toolkit for users to work with:
   This means that newer, faster means of scientific programs can be written with Octave, similar to other scripting languages that have taken advantage of GTK. More over, people could do faster prototyping with interpreted languages, and Octave-GTK can become an Ideal RAD tool for academics, and developers alike.

2. It is possible to write a GUI for Octave with Octave itself:
   This is indeed an elegant solution, but we cannot comment till the performance-time trade off matches user satisfaction, but however it is still a possibility.

3. The GNOME connection:
   GNOME technologies like Bonobo, libgnomeui could further be ported to the Octave language, making Octave powerful like some general purpose programming language. Unlikely, but possible. We could write components for GNOME from Octave given this set of primitives.

4. Library Reuse:
   An Octave library that offers, so much computational routines, could well be used for scientific computing, rather than re-implement it. Similarly, it is unnecessary to rewrite a GUI toolkit for Octave, when it is possible to use GTK, which dovetails the need. Thus Octave-GTK, will achieve the Library reuse, which simply means lesser code to write, maintain and ship.

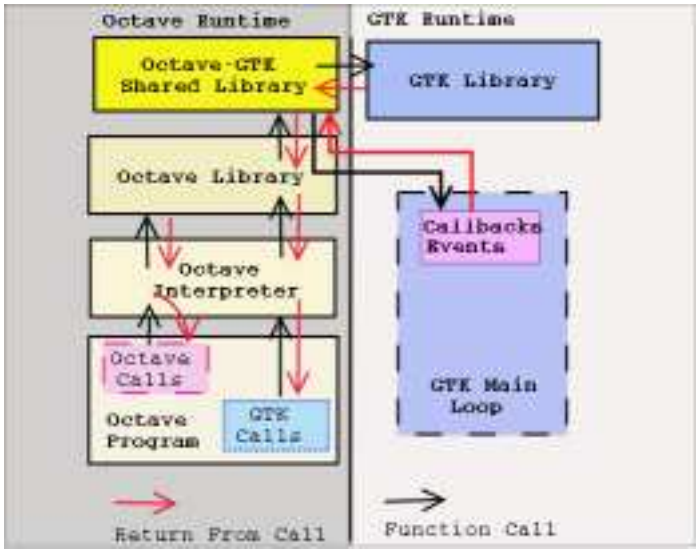

Figure 1: Octave-GTK Architecture

7 Conclusion

Using Octave-Gtk it will be possible to have simpler API's for scientific computing and and this will significantly reduce the effort in GUI development. Moreover Octave itself will empowered to compete with proprietary alternatives. Octave-GTK will come into being, and introduce a powerful, and free alternative.

8 References

1. www.gtk.org, GTK sources, documentation, mailing lists [gtk-app-devel@lists.gnome.org] and architecture.

2. John W Eaton created GNU Octave. www.octave.org, Octave sources, documentation, Manual, mailing lists[graphics@octave.org]

3. Octave-GTK project http://octave-gtk.sourceforge.net/

4. .Paul Kienzile maintains Octave Forge at http://octave.sourceforge.net

5. www.gnu.org GNUś Not Unix is the project aimed to make a free Unix like operating system. Also GNU/Linux

6. www.pygtk.org , James Henstridge's pygtk binding code,code generator, and design documentataion.

7. SLang GTK binding by Michael S. Noble, binding code, documentation on code generator, design. http://space.mit.edu/ mnoble/slgtk

8. www.boutell.com/gd, GD, Graphics drawing library for PNG,JPEG,GIF,WBMP images

9. gd-Octave, A port of the GD library for GNU Octave, by Muthiah Annamalai, Hemant Kumar http://freshmeat.net/projects/gdoctave

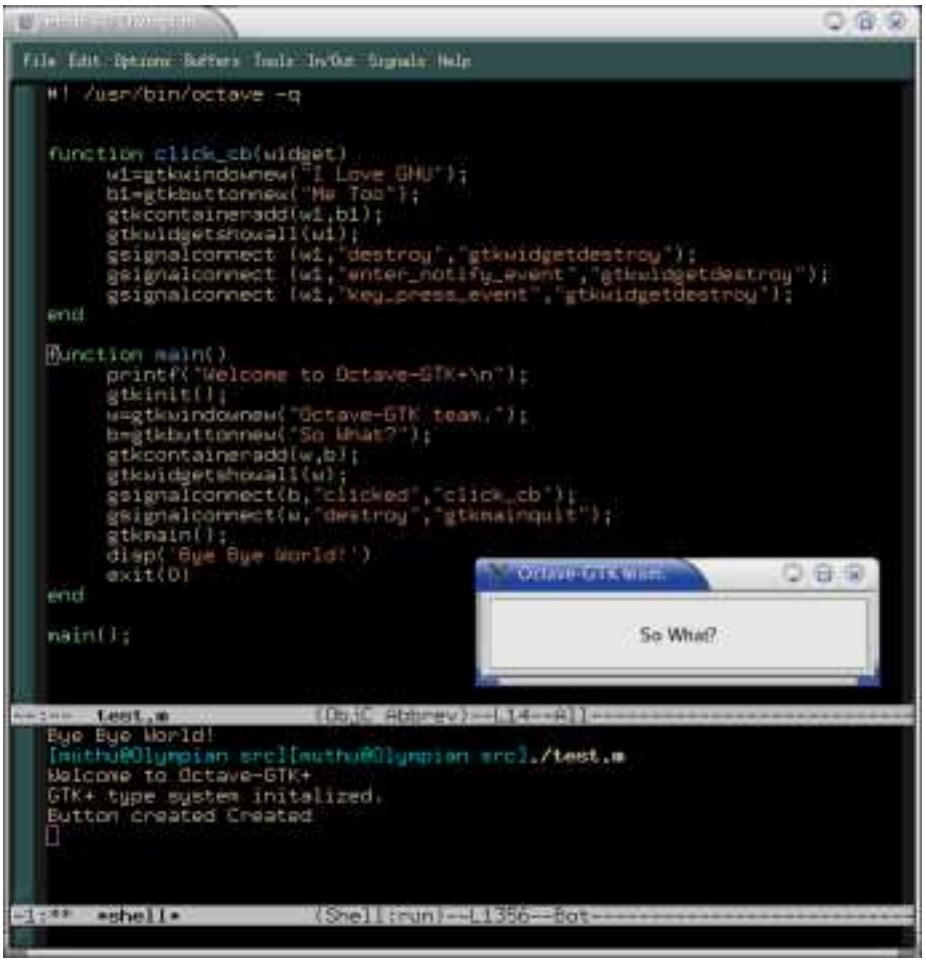

Figure 2: Octave-GTK Prototype