A Succinct Graphical User Interface Programming Model for Low-end Embedded Devices

Masakazu Yoshida
AXIS, Inc.
Nara-shi, Nara, Japan
Email: masa@computer.org

***Abstract*—Adapting graphical user interface, or GUI, is a promising way for low-end embedded devices to accommodate ever-increasing functions within limited space and to make them accessible for users. However, most of existing GUI tool kits are not suitable for this purpose because their intended targets are desktop computer applications, not embedded systems. Since the user interaction model and resource restrictions of low-end embedded devices are totally different from those of desktop applications, we propose a succinct GUI programming model as an alternative to the one for desktop computers. The proposed model mainly addresses three design issues: conforming to the appropriate interaction model, resource-conscience, and adaptation to crude run-time environments. We implemented MGT, an experimental GUI toolkit based on the proposed programming model, to evaluate the effectiveness of our model.**

I. Introduction

Graphical User Interface, or GUI, can be a promising way for low-end embedded devices, such as remote controllers, to accommodate their ever-increasing functions. Also adaption of GUI makes it possible for those devices to multiplex their functions into some groups and allow users to painlessly navigate through them.

While GUI tool kits are certainly necessary to develop GUI applications effectively, there are not any tool kits suitable for low-end embedded devices. Most of existing GUI tool kits are primarily designed to develop desktop applications, and other tool kits aiming for embedded devices, in fact, employ the similar software architecture of the former. In comparison with desktop systems, low-end embedded devices have extremely restricted resources. Typical microprocessors used in such devices have less than 64K bytes of RAM so that it is almost impossible to adapt desktop GUI tool kits.

In this paper, we present a succinct graphical user interface programming model specialized for low-end embedded devices. The design goals of our model are the followings:

- Adaptation of the appropriate interaction model
- Resource-conscience
- Flexibility to adapt for crude run-time environments

Firstly, the interaction model of GUI for low-end embedded devices is significantly different for the one of GUI applications running on desktop systems. The latter is more flexible and dynamtic in behaviour, but those properties are redundant because it is common for low-end devices that there is a single application exclusively occupying the screen.

Secondly, on low-end embedded systems, the scarcest resouce is RAM. In some cases, there is barely sufficent RAM for the frame buffer. The design of tool kits ought to be aware of this.

Thirdly, tool kits ought to be able to adapt for a vast range of system configurations. Since run-time environments of embedded systems are wildly diverse, even you can not assume any kinds of operating systems.

II. Proposed Programming Model

To achive the design goals described earlier, we devise a succinct GUI programming model for low-end embedded devices.

A. Single Thread of Fixed Size Windows

In our programming model, GUI should be presented as a thread of fixed size windows. Although desktop GUI tool kits employ the mutliple windows model that allows the user to handle multiple applications simultaneously on the screen [1, pp. 435–469], our model adapts the single window model that presents the user single window at a time. This implies that there is no need for the window manager nor dyamic layout facility because there is only single application and the size of the windows is fixed.

B. Static Behavior

In our programming model, the tool kit does not provide almost any kinds of dynamic behavior. In case the size of windows is fixed and UI entities, such as windows and controls, are laid out in advance, there is no need for those entities to be dynamic. Therefore most of the design time properties of UI entities, such as position, size, and background color, can be located in ROM.

C. No Call-Back Functions

In our programming model, UI entities do not have any call-back functions at all. To support call-back functions is very expensive because it requires considerable amount of RAM for every UI entity that has any call-back function properties. Since the nature of low-end embedded devices' GUI is mostly static, it is rare that we have to do something with call-back functions.

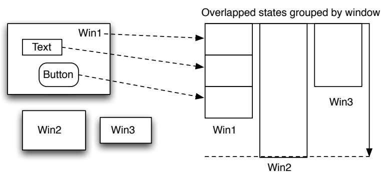
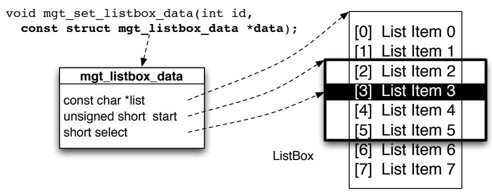


Fig. 1. Overlapped states grouped by window



```
void mgt_set_listbox_data(int id,
  const struct mgt_listbox_data *data);
```

Fig. 2. ListBox accepts intructions supplied by application

D. *Limited Life-Time of States*

In our programming model, the states of UI entities have limited life-time so that it is possible to overlay RAM area for the UI entities grouped by window (Fig. 1). Since their states are only valid when their parent window is active, the states are reset every time when the window is going to be activate or reactivate.

E. *View Functionality Only*

In our programming model, UI entities should provide *View* functionality only, in contrast to MVC (Model-View-Controller) design pattern [2, pp. 123], in order to minimize RAM requirement. Therefore the application is responsible for what and when to display on UI entities. For example, the ListBox control of our experimental implementation, described in the later section, passively accepts instructions from the application (Fig. 2). This implies that the control flow in our model is a reversal of the one in desktop GUI applications.

F. *Tailoring Your Own Event-Main-Loop*

In our programming model, the application should take control of the event main loop, because, on crude run-time environments, it is impractical to assume any sorts of portable software abstractions, such as an operating system. So tool kits merely provide building block fictionalises:

- dispatching an input event
- redrawing the frame buffer if needed

III. Experimental Implimentation

Based on the proposed programming model described in the previous section, we implemented a production strength GUI tool kit, called MGT – *Minimal GUI tool kit*, specialized for low-end embedded devices. MGT is entirely written in ANSI standard C language and provides seven UI entities: Form (Window), Label, Button, CheckBox, IconButton, ListBox, and PictureBox. MGT is designed to be used in the combination of a program generator that produces C source program

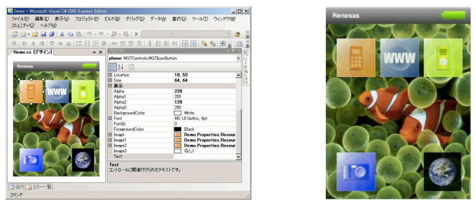


Fig. 3. Designing GUI with Visual Studio (left) and screen-shot (right)

```
#include "mgt.h"
static unsigned short framebuff[320 * 240];
MGT_Event *poll_event(void);

void main(void)
{
  mgt_init();
  mgt_flip(framebuff, 320, 240, 640);
  while (1) {
    mgt_tick(poll_event());
    /* delay for periodic repetation */
  }
}

void mgt_hook(int type, int obj_id)
{
  /* handle internal events here if needed */
}
```

Fig. 4. Sample main program

TABLE I
The footprint of MGT (with i686-apple-darwin9-gcc-4.0.1)

Memory Requirements	Size (Bytes)
ROM used by MGT library	10,220
ROM used per UI entity	60
RAM used by MGT library	336
RAM used per UI entity	8

from design data created in the manner of WYSIWYG with Microsoft Visual Studio (Fig. 3). Since the portion concerning GUI design is generated automatically, the main program of an application looks something like in Fig. 4.

As a result of adaption of proposed programming model, we can shirk the footprint of MGT drastically as shown in TABLE I. This small footprint [1] make it possible to apply MGT to the extent of 16 bits microprocessors.

IV. Conclusion

We clarify three design goals that GUI tool kits for low-end embedded devices ought to satisfy and propose a succinct GUI programming model to achieve them. Our MGT, a GUI tool kit implementation based on proposed programming model shows that our model is effective to shrink the footprint. We are convinced that it is effective to address a succinct GUI programming model for embedded devices, especially low-end ones.

References

[1] James D. Foley et al. *Computer Graphics: Principles and Practice*. Addison-Wesley, 2nd edition, 1996.
[2] Aaron Hillegass. *Cocoa Programming for Mac OS X*. Addison-Wesley, 3rd edition, 2008.

[1]The footprint includes neither 2D graphics library nor fonts.