Solution of UI based on the Large-Scale Embedded System

Wang Tiantian and Li Honglian*

Department of Electronic Information and Engineering, Beijing Information Science & Technology University, Beijing, 100103, China

Abstract: This article aims to provide a solution of UI (User Interface) of large-scale embedded systems which depends on the PEG library. UI implementation of embedded systems requires a good portability, that no matter what kind of target hardware, it does not require any software components with graphical output capability. PEG is abbreviation of portable embedded GUI, which is a class library. As long as our environment with a C++ compiler and hardware which can output graphics, you can run PEG, and this precisely is the most important point of the development of embedded system. This article describes the PEG application in the embedded system, and aim to the large-scale embedded systems we provides a solution, make the system has good portability, stability and maintainability.

Keywords: Embedded system, PEG, portable embedded GUI, UI solution.

1. EMBEDDED SYSTEMS AND PEG

With the rapid development of science and technology, embedded systems are being widely used in various industries and fields of life. In the development of embedded systems, the ultimate goal is a user-friendly system which can make the human-computer interaction more efficient and productive. The UI (user interface) of embedded systems enables the users in achieving real-time output with good portability. The PEG can reflect these characteristics, irrespective of the kind of target hardware. It does not require any software component as well as it demonstrate graphical output capabilities, substantially reduce the burden on the hardware, saving the cost of embedded systems, and can be flexibly used in various systems.

PEG is abbreviation of portable embedded GUI, which is a class library. The PEG class library provides building blocks for a powerful and extensible graphical user interface. PEG is a library that is fully capable of supporting all of the advanced GUI features we need today, while also accommodating future enhancements.

Due to the embedded system hardware constraints, such as limited memory, memory recall mechanism is particularly important [1]. PEG provides garbage collection mechanism to efficiently ensure the memory is not leaked. The human-computer interaction design need a lot of pictures and text, in embedded systems, we call these pictures and text as resource.

For large-scale embedded systems, how to ensure good management of resources, as well as system maintainability and scalability, are we want to study. In this paper, we propose a solution of UI for the large-scale embedded system design [2].

*Address correspondence to these authors at the Department of electronic information and engineering, Beijing Information Science & Technology University, Beijing, China, 100103; E-mails: wangtiantian109@gmail.com and lihonglian@bistu.com.cn

2. BASIC WORKING MECHANISM OF PEG

2.1. Widgets of PEG

Since PEG is a class library for human-computer interaction, hence the widget library is the core part of PEG. PEG provides a set of widgets for our use containing a button, label, radio button and text box (Fig. **1**). These widgets are almost always inherit PegThing, including our most commonly used operations and needs.

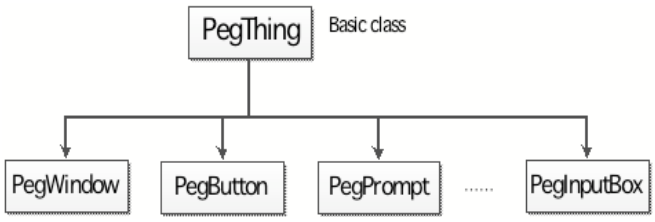


Fig. (1). Relationship of widgets.

PegThing is the base class from which all viewable PEG objects are derived. While you may never create an instance of an actual PegThing in your application, it is very possible that you will derive your own custom control types from PegThing. In any event, every window and control you use is based on the PegThing, thus you will be using the public functions of PegThing when programming with PEG.

A basic precept in the design of PEG is that all graphical objects, from the most complex tabbed notebook or table to the simplest bitmapped button, share a small but significant set of properties. Some of these basic properties include: whether or not the object is visible; if the object has a parent

1874-110X/14 2013 Bentham Open

and who that parent is; if the object has children and who those children are; if the user should be allowed to interact with an object. These and other properties define how each object will participate in your graphical presentation (Figs. **2** and **3**). Class PegThing maintains this information about each PEG object.

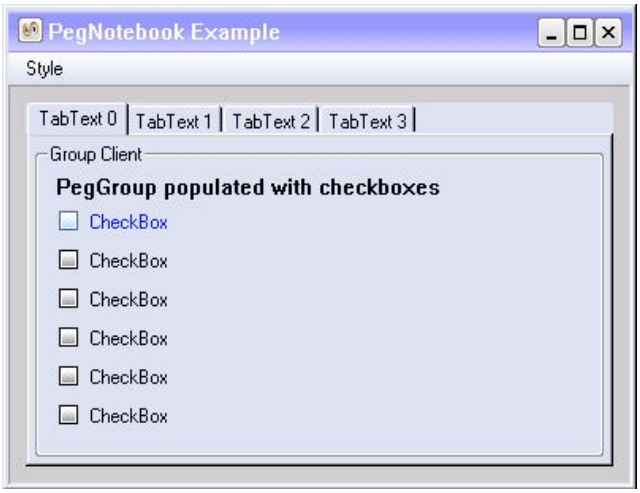
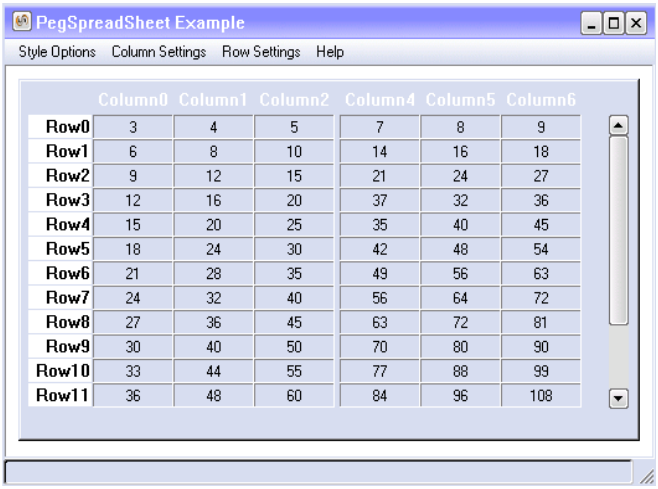


Fig. (2). Example of check boxes.



	Column0	Column1	Column2	Column4	Column5	Column6
Row0	3	4	5	7	8	9
Row1	6	8	10	14	16	18
Row2	9	12	15	21	24	27
Row3	12	16	20	37	32	36
Row4	15	20	25	35	40	45
Row5	18	24	30	42	48	54
Row6	21	28	35	49	56	63
Row7	24	32	40	56	64	72
Row8	27	36	45	63	72	81
Row9	30	40	50	70	80	90
Row10	33	44	55	77	88	99
Row11	36	48	60	84	96	108

Fig. (3). Example of spread sheet.

Widgets are all inherited PegThing, through overriding the Draw() function to achieve what you need to draw, through overriding the Message() function to define a certain message that you need your program to do something. It is this mechanism that allows us to customize for our needs and style of their own parts.

2.2. About Portability

We hope that applications or device drivers can be well compatible in different plat-forms [3], which requires PEG to take transplantation into account. In different operating systems, the word length of the basic data type may not be the same, so in PEG library, basic data types are redefined. In the programming, you need to replace the corresponding data types, such as we have to declare an integer, should be written `PEGINT num = 0;` declare a Boolean type: `PEGBOOL is Right = TRUE;`

The following simple data types are used instead of the intrinsic data types defined by the compiler to avoid conflicts when running on CPUs with differing basic word length and data manipulation capabilities. In all cases, longer bit length types on those machines that do not accommodate 8 or 16 bit data values may replace shorter bit length types. The following definitions, contained in the file pegtypes.hpp, may need to be modified [4] to match the word length of your target CPU. The comment next to each data type describes the storage requirements PEG requires for each type:

```
PEGBYTE signed 8-bit value
PEGUBYTE unsigned 8-bit value
PEGINT signed native int (size unspecified)
PEGUINT unsigned native int (size
unspecified)
PEGSHORT signed 16-bit value
PEGUSHORT unsigned 16-bit value
PEGLONG signed 32 bit value
PEGULONG unsigned 32 bit value
PEGCHAR 8 or 16 bit character storage type
PEGBOOL TRUE/FALSE value
```

2.3. About Real-Time

PegMessage is a data structure used to send and receive messages. PegMessageQueue is the coordinator of message transport in your PEG application. PegMessageQueue is a simple encapsulated FIFO message queue with member functions for queue management. PegMessageQueue provides functions for sending and receiving PegMessage formatted messages. PegMessageQueue also performs timer maintenance and miscellaneous housekeeping duties.

PEG uses the message management mechanism to achieve real-time. When user interacts with the machine, PEG send the user's actions in the form of message, and the message queue will process it by sending this message to a widget which is needed to response. For example, the most common situation is when a user clicks button. When a user clicks a button, it will generate a corresponding message, this message includes the source widget which sends this message, the target widget which needs to receive this message, as well as some of the necessary parameters, and then the message queue will distribute it according to the information [5].

How do messages get into the PegMessageQueue? These are placed in the message queue from one of the three sources:

- Input devices, such as a mouse, touch screen, or keyboard.
- Any other task in the multitasking system.
- From PEG objects themselves.

The messages placed in PegMessageQueue are the driving force behind the graphical interface. These messages contain notifications and commands which cause the

graphical elements to redraw themselves, remove themselves from the screen, resize themselves, or perform any number of various other tasks. Messages can also be user-defined, allowing you to send and receive a nearly unlimited number of messages whose meaning is defined by you. For example, it would be very common to have a graphical element send a message to another task in the system requesting data for display. The target task receives the request and responds, sending the response message back to the graphical element. [6].

PEG defines its own message format. The PEG message format never changes from one operating system to another or when running on the desktop vs. running on your target. When running with a real-time operating system, PEG implements the PegMessageQueue by utilizing the underlying operating system services. To your application level software, it always appears simply as PEG messages running through the PegMessageQueue, regardless of the underlying implementation. This of course helps to make your application software completely portable across operating systems.

Integrated versions of PEG provide PegMessageQueue functionality based com-pletely on the underlying RTOS message services. This implementation is invisible to the external system software, allowing PEG applications to be fully portable between development and real-time environments.

The following example creates and sends a new PegMessage. The message will cause the target object to resize.

```
void SomeObject::ResizeWindow(PegWindow
*pTarget, PegRect NewSize)
{
  PegMessage NewMessage(pTarget, NewSize);
  MessageQueue()->Push(NewMessage);
}
```

The following example window creates a periodic timer when the window is made visible, receives periodic timer messages, and destroys the timer when the window is hidden:

```
MyWindow::Message(const PegMessage &Mesg)
{
  switch(Mesg.Type)
  {
    case PM_SHOW:
      SetTimer(TIMER_1, 100, 100);
      PegWindow::Message(Mesg);
      break;
    case PM_HIDE:
      KillTimer(TIMER_1);
      PegWindow::Message(Mesg);
      break;
    default:
      return PegWindow::Message(Mesg);
  }
  return 0;
}
```

3. ADVANTAGES OF WORK ON THE LARGE-SCALE EMBEDDED SYSTEM

In the face of large-scale embedded systems, such as complex industrial embedded applications, PEG can highlight some of its advantages. On the one hand, PEG has its own working mechanism for making our program structure clearer and easier to manage. On the other hand, the mechanism of processing resources that contains the text and pictures determines [7] when the program is large-scale those resources will also be well managed.

3.1. Advantages of Programing Mechanism

3.1.1. Parent, Child, Sibling

These terms refer to the relationship between the windows, controls, and other items that are all part of your interface. A control that is attached to a window is termed a Child of that window. Likewise, the window that contains the control is termed the Parent window. If there are several controls attached to the same window, those controls refer to each other as siblings.

While we have just described the most common case, there is nothing internal to PEG that prevents a window class such as PegWindow from being the child of a control, such as a PegButton. In fact, it is often very useful to construct custom objects using exactly this type of parent-child relationship.

Some GUI platforms place restrictions on the number of parent-child generations that can be nested within the same window, or even within a single application. PEG imposes no such restrictions, nor will anything prevent an object that is a parent ob-ject in one case from becoming a child of another object in a different case [8]. This is a powerful feature of PEG, because it allows you to re-use custom objects that you create in a variety of different ways.

3.1.2. Garbage Collection Mechanism

Dependent on the PEG library, we will add some widgets on the window by Add() function, by default those widgets such as buttons are child widgets of the window, and while we destroy the window, the garbage collection mechanism of PEG will make all the widgets which add on the window destroyed out. This can ensure unnecessary memory leak. [9]. We know that in the embedded systems development, especially in some industrial applications, the problem of memory leaking is immeasurable. For example, some machines used in measuring the density or the gas composition, once started, it will not be possible to shut down or restart for ten years, if it has been running a memory leak. It would lead to insufficient system memory collapse after a long run, bringing huge economic losses.

3.2. Management of Resources

The fonts, bitmaps, strings, and colors we utilize in our application are called Resources. In many ways resources are independent of the application. We can change and modify our resources to change the user interface without making any changes to the application software. We can even change

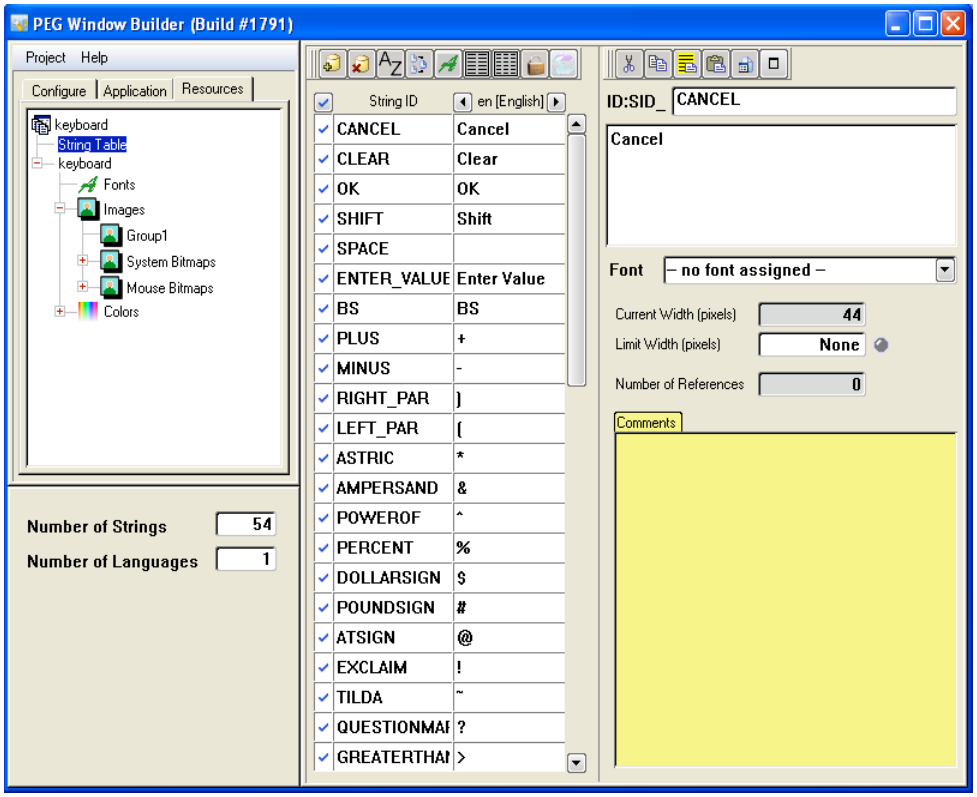


Fig. (4). PEG window builder.

our resources "on the fly" when the system is running. An example of this would be to change the active language or color theme of the application [10].

PegResourceManager manages the system resources. PegResourceManager allows us to add, remove, and modify resources at compile and runtime. Resources are registered with the ResourceManager, where they are assigned a resource ID. The application software always refers to a resource using the resource ID rather than using a direct reference to a font or bitmap or string. This abstraction is what makes it possible to easily modify your resources without requiring any changes to the application software [11].

When the project is large enough, we can still intuitively and objectively manage and control resources by WindowBuilder. WindowBuilder is a graphical user interface tool [12], which allows to type any ID name desired for each instance of each resource type (Fig. **4**).

4. CURRENT & FUTURE DEVELOPMENTS

Since PEG has good portability and real-time, making the PEG library can be applied to UI development of embedded systems.

In addition, the PEG has a special garbage collection mechanism and a unique resource management capacity for large-scale embedded systems, we have therefore suggested a solution depending on a PEG library which demonstrate better performance.

CONFLICT OF INTEREST

The authors confirm that this article content has no conflicts of interest.

ACKNOWLEDGEMENTS

Declared none.

REFERENCES

[1] P. Koopman, "Embedded system design issues (the rest of the story), in *Proc. IEEE Int. Conf. Computer Design. VLSI Computers Processors*, Austin, TX, pp. 310-317, October 1996.

[2] P. Brelet, A. Grasset, P. Bonnot, F. Ieromnimon, D. Kritharidis, and N. Voros, "System level design for embedded reconfigurable systems using morpheus platform," in *[VLSI (ISVLSI), 2010 IEEE Computer Society Annual Symposium on]*, pp. 500-505.

[3] J. Saib and A. Suzuki, "*GUI resource editor for an embedded system*", U.S. Patent 6,429,885. 2002-8-6.

[4] J.A. Morgan, and C. Venkatraman, "*Embedding web access mechanism in an appliance for user interface functions including a web server and web browser*", U.S. Patent 5,956,487. 1999-9-21.

[5] K. Hines, and G. Borriello, "A geographically distributed framework for embedded system design and validation", in *Proceedings of the 35th annual Design Automation Conference.* ACM, 1998: 140-145.

[6] J.G. D'Ambrosio, and X.S. Hu, "Configuration-level hardware/ software partitioning for real-time embedded systems", Hardware/ Software Codesign, 1994, *Proceedings of the 3rd International Workshop on. IEEE*, 1994, pp. 34-41.

[7] F. Slomka, M. Dörfel, R. Münzenberger and R. Hofmann, "Hardware/software codesign and rapid prototyping of embedded systems", *IEEE J. Des. Test. Comput.*, vol. 17, no. 2, 2000, pp. 28-38.

[8] S.R. Lewallen, "*Method, system, and program for generating a graphical user interface window for an application program*", U.S. Patent 6,801,224. 2004-10-5.

[9] G. Panarello, J.W. Bermann, A. Koifman, and J.C. Scano, "*Platform independent enhanced help system for an internet enabled embedded system*", U.S. Patent 6,289,370[P]. 2001-9-11.

[10] R.J. Wolf, "*User interface with embedded objects for personal computers and the like*: U.S. Patent 5,838,321[P]. 1998-11-17.

[11] A.R. Allouche, "Gabedit-a graphical user interface for computational chemistry softwares". *J. Comput. Chem.*, vol. 32, no. 1, 2011, pp. 174-182.

[12] L.J. Farrugia, "ORTEP-3 for Windows-a version of ORTEP-III with a Graphical User Interface (GUI)", *J. Appl. Crystallogr.*, vol. 30, no. 5, 1997, pp. 565-565.

Received: July 23, 2014 Revised: August 13, 2014 Accepted: August 16, 2014