

Exhibit 1022

SHADOW: A Cross-Platform GUI Middleware for Home Devices

Bomjun Kwon, Sungho Yoon, Youngki Ahn, Hyung-Won Jung, Jangseok Seo
Software Laboratories

Samsung Electronics Co., Ltd.

Suwon-si, Gyeonggi-do, 443-742, Republic of Korea

{bomjun,sh73.yoon,ykahn,won.jung,jseo}@samsung.com

Abstract

SHADOW is a middleware for building GUI applications in home digital media devices. It is a part of home software platform that solves portability issues and it provides a universal application framework for OS independence. It also supports C++ class libraries for object-oriented design of GUI applications. SHADOW was successfully used for building GUI applications for TV-like home devices. In this paper, we describe how SHADOW is designed and implemented in order to support home applications.

1 Introduction

The features of CE products are getting more complex and diverse in order to live up to customers' expectations. Emerging digital media technologies have enabled people to deal with various kinds of digital media including images, movies and music. These technologies are realized by newly developed CE products, such as digital televisions, digital cameras, and MP3 players, that have never existed a few years ago. These products do not provide only simple and essential features for creating and playing digital media, but also introduce more powerful features for editing, sharing, and managing digital media.

At the first presentation of the new CE products, people were only interested in *what they can do with them*. There were so many features that people were just happy with them. But nowadays, they are more interested in *how they can do it* after realizing that those features cannot be efficiently used without some nice user interfaces. Now they know they cannot do anything easily with a TV remote controller that has more than one hundred of buttons. More simple and useful guides that are displayed graphically on a screen are preferable. This makes GUIs on CE devices more popular nowadays.

Building GUI applications for CE devices are not so simple; it is accompanied by several problems about software

development on CE devices. Although the GUIs can be very similar among the same kind of products, they may differ in hardware architectures, device drivers, and operating systems. This causes a portability issue. Some of essential GUI applications had to be re-implemented just because the development environment of a new product, such as graphics chipset and operating system, is changed. Lacks of advanced GUI features in embedded graphics libraries made GUI application development harder, too. Primitive graphics APIs that were provided by hardware vendors did not satisfied the GUI application developers. They were competing with fancy applications based on PC's desktop window system, with which customers experienced high-quality GUI.

We have been trying to find good solutions for developing GUI applications of CE devices. Surely, there are many solutions that supports embedded system GUIs [1, 2, 3, 4]. However, it was hard to find one that meets the critical requirements for CE GUI solution, such as OS/architecture independence, high level GUI objects, and reusable component framework. Some of them were not supporting all of three major operating systems for embedded systems, that is, Linux and VxWorks, and Microsoft Windows [5]. Some of them were providing GUI objects that mostly rely upon 2D positioning input devices, such as mice, touch-screens and styluses. They may be useful for desktop or personal devices, but they are not usual interfaces for home devices yet. Some of them had only C APIs, which were not easy to extend comparing to C++ class APIs. We also needed our own proprietary solution that avoids license issues. Thus, we started to develop a novel GUI middleware that can be used for developing GUI applications in CE devices, especially for home devices including digital televisions, set-top boxes, and DVD recorder/players.

SHADOW is the result of our effort to make a middleware for building GUI applications in home digital media devices. It is an abbreviated word of *Samsung Home-platform Application-framework Device-independent Object-oriented Workspace*. Although this

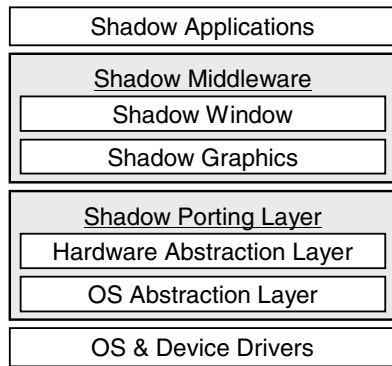


Figure 1. Shadow architecture

seems to be unnatural interpretation of a very common word, it enables us to quickly guess the important features of SHADOW: it is a part of home software platform that solves portability issues and it provides a universal application framework for OS independence. It also supports C++ class libraries for object-oriented design of GUI applications.

Currently, a good many kinds of digital televisions, that are newly developed in Samsung Electronics, are filled with SHADOW applications. This demonstrates that SHADOW can be successfully used for building GUI applications for TV-like home devices. In this paper, we describe how SHADOW is designed and implemented in order to support home applications.

The rest of this paper is organized as follows. Section 2 describes the architecture of SHADOW in detail. Section 3 provides brief discussion on design and implementation issues about SHADOW. Section 4 shows the result of application and the current status of related work. Finally, Section 5 concludes the paper with future directions.

2 Architecture

SHADOW is developed as a software platform for home applications. That is, it supports an OS and hardware independent environment for developing applications. Figure 1 shows the architecture of SHADOW. The two main layers of SHADOW, SHADOW Porting Layer and SHADOW Middleware, provide compact but useful libraries for building GUI applications.

2.1 Shadow Porting Layer

SHADOW Porting Layer abstracts the functions of the hardware and OS, so that middleware components and applications on top of this layer can be free from portability issues. SHADOW Porting Layer has two components:

OS Abstraction Layer (OSAL) and Hardware Abstraction Layer (HAL).

OS Abstraction Layer

The classes of OSAL provides programming interfaces, which let a software developer use OS functions in a unique way. Every component that are developed using OSAL become OS independent, and they can be reused without considering portability problem. OSAL classes can be classified into two categories: low level abstraction classes and high level abstraction classes. The low level abstraction classes simply wrap the functions of OS or C libraries. Memory allocation, error message management, string operation, and math functions are typical examples of low level abstraction classes. The high level abstraction classes are more complex than the low ones; they provide task/thread management, and inter-task communication including event sending and handling. This is a kind of application framework, which guides how to build a portable application in a recommended way.

Hardware Abstraction Layer

HAL is mainly used for abstracting I/O devices, in particular, graphics display devices. It manages attributes for graphics devices and gives a common way to access the frame buffer of the video memory. HAL also provide a way to directly use the hardware accelerated graphical operations available for some graphics hardware.

2.2 Shadow Middleware

SHADOW Middleware provides various kinds of high level GUI functions that are based on the low level functions of SHADOW Porting Layer. Each component has several C++ classes that enables the application programmers to easily extend the base function of SHADOW and design their applications in object-oriented fashion.

Graphics Library

Graphics Library provides many low level graphics operations such as primitive drawing, text drawing, bitmap image decoding and displaying. It supports graphics context for managing screen regions and drawing attributes. Virtual coordinates system makes GUI application to be designed at a fixed logical screen resolution.

Window Library

Window Library uses the low level graphics operation provided by Graphics Library in order to implement GUI objects, which is called widgets, that are directly manipulated



Figure 2. Sample GUI layout

by end-users. An application usually assembles these widgets to make a GUI scene that is presented to end-users. Figure 2 shows an example GUI layout that exploits several widgets: label, button, list, scroll bar, and frame widgets.

3 Design and Implementation Issues

In this section, we describe several issues about SHADOW design and implementation. Unlike developing an ordinary desktop window system, SHADOW faced several technical problems that were critical to be a part of commercial CE products.

3.1 Supporting Multiple OS's

The minimal set of OS's supported by SHADOW are Linux [6], VxWorks [7], and Microsoft Windows [8]. Linux and VxWorks are the two main embedded OS's that are used for home digital media devices in Samsung Electronics, and Windows is the most popular OS for application developers. We thought that Windows must be supported no less than the other popular embedded OS's. This led to a great welcome from the SHADOW application developers because most of their jobs could be done purely in their PCs, including debugging and analyzing their programs, and without any modification of the code, their programs were executed in target devices well.

To support multiple OS's, OSAL was carefully designed not to have a feature that cannot be realized in some OS's. Of course, there are drawbacks giving up some nice features that are supported by only some outstanding OS's. However, our first goal was to make application portable at all, and this means we are allowed to use the only features available for all OS's we are targeting.

Regarding this issue, one of the most important features of SHADOW OSAL became its task model. SHADOW is one-process multi-thread model, that is, there is only one SHADOW process and every SHADOW applications are ex-

ecuted as a set of threads belonging to this process. This is because the process model of Linux cannot fit into some embedded OS's including VxWorks. The major three OS's supports almost similar thread model. Thus, we designed SHADOW to support only thread-based task management.

We recommend application developers not to directly use any of system calls or external library supported by OS. For example, even including standard header files such as `stdlib.h` is prohibited. They can only include SHADOW headers, which also includes minimal but complete set of classes supporting general system calls and C/C++ library functions. By doing this, SHADOW applications can be totally free from any of OS-related features and no modification is needed to port them to other OS environment.

3.2 Font Engines

Text drawing is one of important graphics operations that are very frequently used. The quality and time performance of text drawing is very critical since most of GUIs have so many text strings and end-users are always reading them. SHADOW supports two kinds of fonts: grey-scale bitmap fonts and scalable vector fonts. They have advantages and disadvantages each other. The bitmap fonts are very easy and efficient to draw, but we cannot manipulate them maintaining the original visual quality. The vector fonts can be freely scaled, rotated, and transformed to satisfy designer's requirement without loss of visual quality, but to draw them, it requires heavy and complex font engines that requires more calculation time.

The bitmap font is drawn by built-in SHADOW Bitmap Font Engine, which is tiny and always included in Graphics Library. For vector fonts, we decided not to include any of font engines by default. Instead, we selected some popular font engines that meet designer and application developer's expectation and provided consistent class interface to use them. Currently we are supporting Bitstream's Font Fusion [9] and FreeType [10]. A system integrator can select an option to link SHADOW and external vector font engines



Figure 3. Font drawing samples

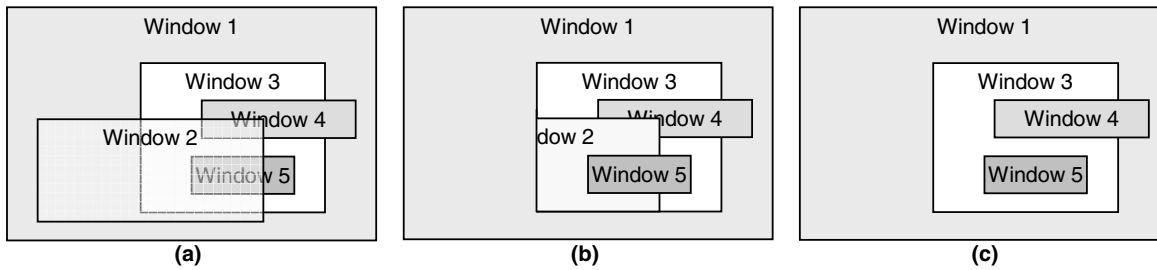


Figure 4. Abnormal intermediate layout of windows

when he compile SHADOW for his target.

Fonts are loaded at run-time, so they can be changed dynamically for altering look-and-feel of the GUI. Various kinds of vector fonts can be used as long as they are supported by external vector font engines, Font Fusion and FreeType. Most of popular font formats are supported: TrueType, PFR, OpenType, PCF, Type1, and so on. Figure 3 shows selected vector font operations.

3.3 Window Expose Operation

Window system of SHADOW is very similar to ordinary desktop window system, such as Windows and X-Window. It supports window hierarchy and full depth clipping. SHADOW adopted an event-driven communication for handling messages between GUI objects. SHADOW windows are also getting several kinds of events to process. They are usually input key signals and expose events.

An expose event is sent to a window when part of the window must be drawn again. This happens frequently by several reason: some obscuring window is destroyed or moved away, the window is just shown from hidden status, or that region is continuously updated by an animation thread, etc.

A problem with these expose events is that a single manipulation of window can result multiple sequence of expose events and they are not processed in the same time. Figure 4 shows a typical example of this problem. In Figure 4-(a), Window 2 is blocking all other windows and is about to be destroyed. When Window 2 is destroyed, Window 1, 3, 4, and 5 get expose events for them to draw their contents covered by Window 2 so far. Since expose event is triggered for each window, at least four expose events are sent to the windows. In fact, Window 3 has a complex-shaped expose region so it may be divided into several rectangular expose events, resulting more expose events. The expose events are processed in random sequence because the windows are possibly managed by multiple threads. Even there is a unique sequence of event processing, they are not processed at once. So they show some intermediate status like Figure 4-(b). At this time, Window 1 and 5 has finished pro-

cessing their expose events, but the others did not yet. After all, when all expose events are processed, the abnormal layout is cleaned up and finally we get Figure 4-(c).

This happens everyday on our desktop window system. But normally it is done in less than 50 milliseconds and no one complains about that. However, in CE products, this maybe a significant problem. Usually CE devices have less powerful CPUs and hardware comparing to the desktop PC systems. So this kind of odd layout is displayed for a longer time, and the users can be annoyed by this broken graphics.

In order to overcome this problem, we adopted double buffering [11]. The results of expose event processing are accumulated in an off-screen buffer until all of the expose events are processed. After that, the contents of the off-screen buffer are copied to the frame buffer at once. Though this causes latency of response to user input and slightly more processing time for copying from back to front buffer, it clears away the abnormal intermediate state of expose event processing.

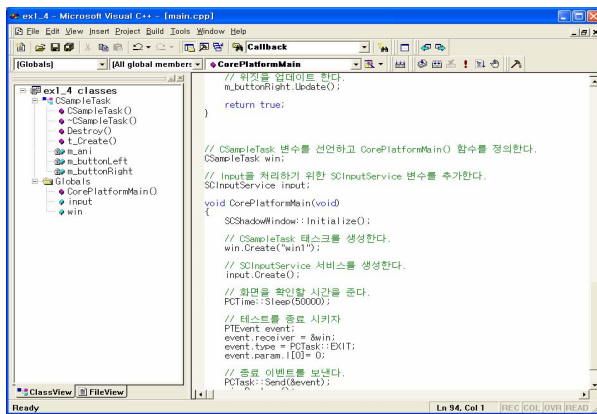
4 Results

SHADOW was implemented for several target systems as well as for Microsoft Windows PC system. In typical Linux target systems, the total size of SHADOW libraries is about 600 to 800 Kbytes. This does not include other external data

Component	Linux/mipsel gcc 3.3.1	Linux/i686 gcc 4.0.1
OS Abstraction	117	97
H/W Abstraction	35	20
Shadow Graphics	354	292
Shadow Window	325	237
Total	831	646

<unit: Kbytes>

Table 1. Library sizes of Shadow components



(a)



(b)

Figure 5. Shadow development environment based on Microsoft Windows PC

such as fonts and bitmap images, which is dependent on application and GUI design. Table 1 shows the size of each component libraries. In Linux, These libraries can be linked to SHADOW applications statically in order to eliminate the unused class code.

As shown in Figure 5-(a), SHADOW developers can use their own PC environment for developing SHADOW applications. They can be executed in the same PC environment that the program was written and compiled. Figure 5-(b) shows a target-emulating front-end of SHADOW, which displays the GUIs and accepts user's keyboard input.

The background image of the emulation front-end is just a given bitmap image or animation. Currently, we are working to emulate television channels utilizing digital tuner attached to the PC or just giving multi-channel broadcast stream stored in hard disk drives.

5 Conclusions

In this paper, we have described the motivation, the design and the implementation of the SHADOW middleware platform. SHADOW is a total GUI solution for CE devices. It is designed to have object-oriented component architecture, and the abstraction layers enable application developers to write perfectly portable GUI code, which leads to reducing cost and time for product time-to-market.

There are many future directions in which to improve SHADOW. The future version of Shadow will provide more dynamic features including three-dimensional graphics engine, such as OpenGL ES [12], and three-dimensional animated widgets. And we would also like to introduce script engine, which can be used for abstracting and separating widget behavior from the native code, and visual layout editor for WYSIWYG GUI design.

References

- [1] "Embedded Linux Graphics Quick Reference Guide", <http://www.linuxdevices.com/articles/AT9202043619.html>, December 2005.
- [2] Carol Røslund, "Qt/Embedded Overview", <http://www.trolltech.com/products/qt/qt3/embedded/index/>, April 2006.
- [3] Alexander Larsson, "GtkFB: GTK+ for the Linux Framebuffer", <http://www.linuxdevices.com/articles/AT9024868021.html>, March 2001.
- [4] "Zinc — Platform independent library for GUI development", <http://www.psa-software.com/zinc.asp>.
- [5] Jim Turley, "Operating systems on the rise", <http://www.embedded.com/showArticle.jhtml?articleID=187203732>, June 2006.
- [6] *Linux Online!*, <http://www.linux.org/>.
- [7] *VxWorks Programmer's Guide, 5.4*, Wind River Systems, Inc., 1999.
- [8] *Microsoft Windows Family Home Page*, <http://www.microsoft.com/windows/default.mspx>.
- [9] *Font Fusion Overview*, http://www.bitstream.com/font.rendering/products/font_fusion/ffoverview.html.
- [10] *The FreeType Project*, <http://www.freetype.org/>.
- [11] Sungho Yoon, Bomjun Kwon, and Jangseok Seo, "Apparatus and Method for Displaying Graphic Objects Concurrently", U.S. Patent Application No. 11/231,898.
- [12] *OpenGL ES — The Standard for Embedded Accelerated 3D Graphics*, <http://www.khronos.org/opengles/>.