A GRAPHICAL USER INTERFACE FRAMEWORK FOR DIGITAL TELEVISION

César Pablo and Vuorimaa Petri

Telecommunications Software and Multimedia Laboratory
Helsinki University of Technology
P.O. Box 5400 FI-02015 HUT
FINLAND
Tel: +358 9 4515128
Fax: +358 9 4515351

E-mail: pcesar@tml.hut.fi and Petri.Vuorimaa@hut.fi

ABSTRACT

Currently, the number of non-PC devices used for interactive applications is increasing. Digital television set-top boxes, PDAs, and mobile phones are typical examples. Interactive applications are controlled via a Graphical User Interface (GUI). Consequently, a GUI framework is required. The objective is a unified GUI framework for all these new devices. Since one of the new devices is digital television, an implementation of a digital television GUI framework is presented. It has been developed following the HAVi specifications. Its main characteristics are use of Java, separation of look and feel, and lightweight widgets. The implementation is tested with a digital television application. The main conclusion proposed is that new devices should use a device-oriented GUI framework implementation. Moreover, specific implementations can be obtained by modifying the look or feel of the widgets.

Keywords: Digital television, DVB-MHP, GUI, HAVi, Java.

1. INTRODUCTION

The key idea behind this paper is device-oriented GUI framework implementation. In the future, multiple devices, besides a PC desktop, are able to run multimedia rich applications [Vierinen01]. The final goal is to create a unified GUI framework, which can be modified depending on the device at hand.

Digital television set-top box is an example of the new devices. Digital television refers to the broadcast of the television signal by digital means [Fox98]. The signal is transmitted compressed using digital format. Hence, the bandwidth needed to broadcast drastically decreases [Milenkovic98]. This increment of bandwidth available can be used in different ways (e.g., broadcasting more channels and including new interactive services) [Vuorimaa00].

In the European countries, a common Application Program Interface (API) for digital television, called DVB-MHP, is used. An API provides a platform independent interface between the applications and the underlying system. DVB-MHP has selected Home Audio/Video Interoperability (HAVi) User-Interface as a standard [DVB00].

Peng et al. [Peng01] and Sivaraman et al. [Sivaraman01] have proposed a digital television environment, which is MHP compliant. It includes hardware, system software, a Java Virtual Machine (JVM) and set of APIs needed, an application manager, and three different services covering the service architecture. The current research is the implementation of a GUI framework following HAVi specification.

2. GUI ELEMENTS

GUIs are built around a toolkit of widgets [Olsen98]. These widgets (e.g., Button, Label, or List) are “ready-made” screen items. So, the developer does not have to deal with the underlying system. Her

work mainly consists in the decision of the widgets to use and where to place them in the screen.

GUI widget has two main features, which provide interactivity. The first one is the visual presentation. The second one is the reaction of the widget to the user interaction. When developing GUI toolkits, these are referred as look and feel, respectively.

Widget can be, depending on how it is rendered, heavyweight or lightweight. Heavyweight is one that is associated with its own native screen resources, commonly known as a peer. Lightweight is one that "borrows" the screen resource of an ancestor. Hence, it has not native resource of its own (i.e., it is called peerless).

In digital television environment, HAVi was included in DVB-MHP [DVB00]. HAVi defined HAVi Level 2 User Interface specification, which is a "TV friendly" framework. It uses *java.awt* package as its core. Although most of the classes from this package are not HAVi framework compliant, some of them have been included in the specification (e.g., *java.awt.Component* and *java.awt.Color*) [DVB00].

3. GUI FRAWEWORK IMPLEMENTATION

The framework implemented, called *ftv*, follows HAVi specification. It is a Java lightweight framework based on *java.awt*. In addition, the look and the feel are separated. Moreover, new specific events, forming the *ftv.event* package, are defined.

3.1 Container

Container refers to the place, where components are added. This way, the added widgets become "children" of the container. In *ftv* package, the container, *ftv.FTVScene*, extends *java.awt.Window*. Its main tasks are handling events and painting its child components. Since the widgets are lightweight, rendering is achieved by calling the *paint(Graphics g)* method of each child.

3.2 Look

Look refers to the visual presentation of the widget. Each widget must have an associated look class, which renders it. When the widget is created, a default look class is used. If needed, the developer can associate a specific look class.

In *ftv*, the following default looks are available: animate, graphic, text animate, text, and time. Animate look defines the way to show a sequence of images. Graphic look determines how to show a single image. Text look is used to display text. Text animate look displays a sequence of text. Finally, time look renders a clock.

Developers can also create specific looks depending on their needs. To test this feature, an additional look was implemented, *ftv.FTVGraphicTextLook*. It extends *ftv.FTVGraphicLook*. This new look draws both a background image and a text over it.

3.3 Feel

The feel of a widget denotes its behaviour. Every widget is originated from *ftv.FTVVisible*, which indicates that the component is displayable. Figure 1 is the UML diagram of *ftv.FTVVisible*.

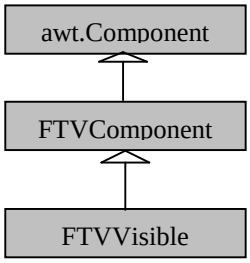

Figure 1. ftv.FTVVisible UML diagram.

In addition, three kinds of behaviours are available: navigable, actionable, and switchable. Navigable widgets are those, which can acquire the focus. Actionable widgets are those, which have an associated functionality. This functionality is launched, when selecting the widget (e.g., button). Switchable widgets are those, which can be selected retaining its internal state information (e.g., toggle button).

3.4 State of the widget

Each widget can reach, depending on its behaviour, up to four different states. These are normal, focused, actioned and actioned focused. When the widget is shown in the screen, it is in a normal state. If the focus is in the widget (i.e., user has navigated over it), it is in a focused state. When the widget has been selected, it is in an actioned state. Finally, when it has been selected and, in addition, gets the focus, it is in an actioned focused state.

3.5 Event handling

Since *ftv* is based on *awt*, it uses delegation event model. Each widget maintains a list of listeners in a class called *ftv.FTVEventMulticaster*. It provides methods, which allow the listeners to add or remove themselves from the list. When an event reaches the widget, it delivers it to the registered listeners.

In *ftv* package, new events were defined, such as *ftv.event.FTVActionEvent*. Hence, new listeners were needed, such as *ftv.event.FTVActionListener*. In addtion, *ftv.event.FTVFocusEvent* includes, besides *awt*'s lost and gained focus, transfer focus (i.e., the focus can be transferred from the current widget to another specific one).

3.6 Toolkit widgets

The developed components can be divided into animate, graphic, and text widgets. Animate refers to those widgets, which make use of a player to sequence the information. This information can be time, text or a set of images. Graphic refers those that only store images. Text defines those that store text.

Animate widgets
Animate widgets, as mentioned above, need a player. The player is a thread, which has some features (e.g. it can be started and stopped as desired, it can be played for a specific number of times, and the speed of the animation can be stated). Figure 2 is the UML diagram of these widgets.

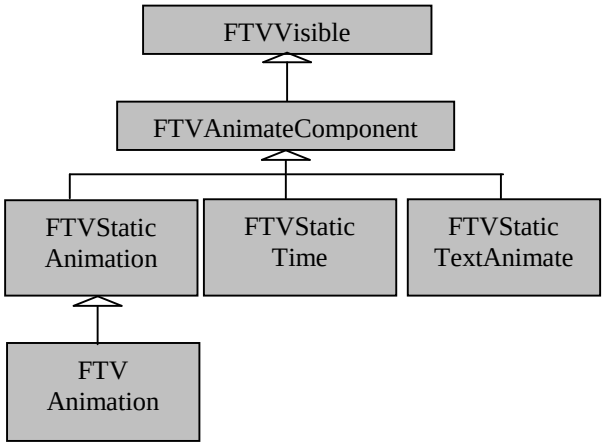

Figure 2. Animate widgets.

Graphic widgets
The graphic widgets are static icon (i.e., static), icon (i.e., navigable), graphic button (i.e., actionable), and toggle button (i.e., switchable). Toggle buttons can be grouped forming a toggle group. This group behaves as a radio buttons group, If decided, the group can, also, behave as checklist.

Text widgets
When creating the widget, the font, the text layout manager and, the background and foreground colour can be selected. The text widget includes static text (i.e., static), text (i.e., navigable), and text button (i.e., actionable).

4. CASE STUDY

As mentioned in section 1, a MHP compliant digital television environment was available before starting the implementation. The objective of this research was to fulfil the lack of a digital television GUI framework implementation. Before it, there were two possibilities. One was to use a general purpose GUI framework, as Swing. The other was to create application specific widgets.

The specific environment covers the following layers. A board PC, 233 MHz processor, 32 MB RAM memory, and 64 MB ROM memory was used as a set-top box. Linux without X-Windows system was the Operating System. The JVM selected was Kaffe. Kaffe uses framebuffer for rendering. An Application Manager controlled all the applications, while the system was running. In addition, three different applications were available, covering different digital television services. These applications comprise a Navigator, a SupertText TV, and an interoperable application. Figure 3 depicts the environment.

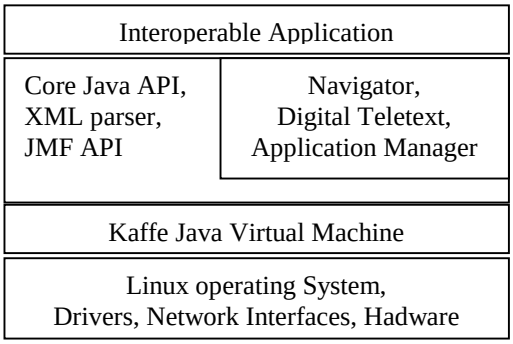

Figure 3. Platform used.

For test purposes, both the Application Manager and the Interoperable Application were used. These two applications were modified to make use of *ftv*. The interactive application was not resident in the set-top box. It was downloaded from Internet. The application manager started this application, when the user pressed a specific key.

The application layout was divided into four differentiate regions. In the top-left, a toggle group, which behaves as a checklist of five toggle buttons, was placed. These toggle buttons did not use the default look. Instead, *ftv.FTVGraphicTextLook* was associated to each. The user could select the service desired via them. In the top-right region, the video was placed. In the bottom-left region, there were two widgets. On the top, a static text indicated the current channel. At the bottom, there was a static time widget. Finally, the bottom-left part informed of the current program by a static text. A screen-shot of the main screen of this application is shown in picture 1.

Picture 1. Main screen of the application.

The *ftv* package has advantages over a general purpose GUI framework implementation. The storage memory is minimised. Also, the widgets are device specific, in this case, television oriented. In the original environment, the containers *used Swing*, which is not needed anymore. Table 1 shows a comparison of storage memory benefits

Package	Storage Memory (KB)
Swing.jar	2420
ftv.jar	96

Table 1. Storage memory of packages.

The *ftv* has, also, benefits over application specific widget implementation. The most obvious is that the widgets can be reused for other applications. In addition, applications are easier to implement, since they do not have to create their own widgets. When a specific look is needed, it can be easily created and associated to an existing widget. Finally, the storage memory of the application is minimised, since its set of widgets it is not included. The interactive application used its own widgets. Table 2 compares the storage memory before and after the use of *ftv* package.

Interactive Application	Storage Memory (KB)
Before using *ftv*	77
After using *ftv*	26

Table 2. Storage memory of Application.

5. CONCLUSION

This paper focuses on the device-oriented GUI framework implementation concept. From now on, a many new devices (e.g., digital television, mobile phone, and PDA) can be used for media rich applications. In this paper, we introduce the idea of device-oriented GUI framework implementation. Since some unnecessary classes for the device at hand can be avoided, the memory consumption is minimised. In addition specific look classes, depending on the device, can be offered.

We used digital television environment as an example of the future devices. A Java lightweight framework for digital television applications, called *ftv* package, was implemented. In addition, an existing environment was used to test this framework. The final result is an application, which uses *ftv*, running on a digital television environment. The final results show how the memory storage was minimised.

6. ACKNOWLEDMENTS

The authors would like to thank the following people. M.Sc. Chengyuan Peng for developing the digital television services used in this paper. M.Sc. Ganesh Sivaraman for creating the system software environment.

REFERENCES

[DVB00] DVB Project Office: DVB: Multimedia Home Platform, *Broadcasting Union*, February 2000.

[Fox98] B. Fox: Digital TV comes down to earth, *IEEE Spectrum*, October 1998.

[Milenkovic98] Milenkovic M.: Delivering interactive services via a digital television infrastructure, *IEEE Multimedia*, Vol. 5, no. 4, Oct./Dec, 1998, pp. 34-43.

[Olsen98] Olsen D. R.: Developing User Interfaces, *Morgan Kaufmann Publishers*, 1998.

[Peng01] Peng C., Cesar P., and Vuorimaa P.: Integration of applications into digital television environment, *in Proc. 2001 Int. Workshop on Multimedia Technology, Architecture, and Application*, Taipei, Taiwan, Sept. 26-28, 2001, pp. 266-272.

[Sivaraman01] Sivaraman G., Cesar P., and Vuorimaa P.: System software for digital television applications, *in Proc. 2001 IEEE Int. Conf. on Multimedia and Expo, ICME2001*, Tokyo, Japan, August 22-25, 2001, pp. 784-787.

[Vierinen01] Vierinen J. and Vuorimaa P.: A browser user interface for digital television, *in Proc. the 9th Int. Conf. in Central Europe on Computer Graphics, Visualization and Computer Vision, WSCG'2001,* Czech Republic, Feb 5 – 9, 2001, pp. 174-181.

[Vuorimaa00] Vuorimaa P.: Digital television service architecture, *in Proc. IEEE International Conference on Multimedia and Expo, ICME2000*, New York City, NY, USA, July 30 – Aug. 2, 2000, pp. 1411-1414.