DEVELOPMENT OF JAVA USER INTERFACE FOR DIGITAL TELEVISION

Chengyuan Peng and Petri Vuorimaa

Telecommunication Software and Multimedia Laboratory,
Helsinki University of Technology,
P.O. Box 5400, FI-02015 HUT, Finland.
pcy@tml.hut.fi and Petri.Vuorimaa@hut.fi

ABSTRACT

The digital television development is one of the most important events in the history of television broadcasting. This paper highlights the user interface issue in digital television environment. We will not discuss the usability of user interface design, but rather the implementation of user interface for *interactive television services*. The background of *Multimedia Home Platform* (MHP) and *Application Programming Interface* (API) is introduced. The paper describes how to develop a Java user interface, which includes not only graphics but also time-based media (e.g., video). Many functions and effects behind the TV visual image have to be implemented. The special features of digital TV user interface are presented by giving an example (i.e., screen information service for ice hockey). Finally, the future possible research topics are briefly addressed.

Keywords: user interface, digital television, Java, application programming interface, interactive television service.

1. INTRODUCTION

In September 1993, European broadcasters, manufacturers, and network operators formed the voluntary *Digital Video Broadcasting* (DVB) group for the delivery of digital television [Fox98]. DVB is the leading standardization groups in digital television. Their objective is to set standards for a TV system driven by commercial market needs. *Multimedia Home Platform* (MHP) is one of the DVB projects. MHP includes set-top-boxes, integrated TV receivers, in-home digital networks, personal computers, network computers, etc [Jacklin97]. A technical group called DVB-TAM (Technical issues Associated with MHP) is working on the specification of the DVB Application Programming Interface (API).

Digital television brings to the TV viewers far more than significantly improved quality of video and audio. It also ushers in the age of true *interactive television*. As the market for digital television grows, content developers are looking for a feature-rich, cost-effective, and reliable *software platform* upon which to build the next generation of *interactive television services* such as Electronic Programming Guide, Video-On-Demand, Enhanced Broadcasting, Multi-camera-angle sporting events, Bill-paying, Home Shopping, Play along with the game show, TV Chat, etc.

The goal for the DVB is to provide an open solution, enabling multiple service providers to operate through a compatible and cost-effective receiver at home. The DVB has decided to use Java as the core specification for the software of the MHP. This is because Java aims to provide a means of implementing applications in a platform independent manner by providing a virtual machine. Java technology is open, scalable, network-aware, portable, and it also supports fast time to market through object-oriented code-reusability [Cornell96]. Some core APIs are included in DVB-Java platform, which is defined in MHP by DVB-TAM.

The user interface of the new interactive television is different from the traditional desktop user interface. Many special challenges arise. Thus our work is aimed at studying and developing the user interface of interactive television services. We are using Java as the programming language of the user interface and digital television as the multimedia home platform.

2. TECHNICAL BACKGROUND

As mentioned in the first chapter, the DVB has chosen Java as its software platform. Also, the DVB has defined some core APIs. The exact meaning of API is described in this chapter. Our Java user interface is based on the several APIs of DVB Java platform. Thus, we'll give a short introduction of Streamed Media API and Java Media Framework (JMF). The Graphical User Interface (GUI) API will be described in detail together with our own research work.

2.1 Application Programming Interface (API)

DVB-TAM has defined an API as a set of high-level functions, data structures, and protocols that represent a standard interface for platform-independent application software [Luetteke98]. It uses object-oriented languages and it enhances the flexibility and re-usability of the platform, as shown in Fig. 1.

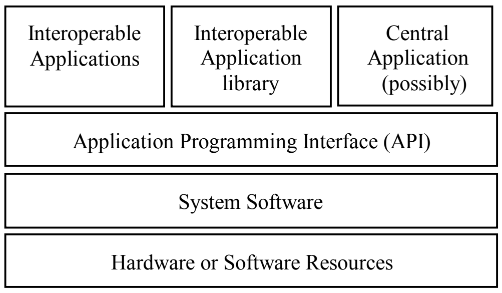

Interoperable Applications
Interoperable Application library
Central Application (possibly)
Application Programming Interface (API)
System Software
Hardware or Software Resources

Fig. 1. DVB TAM Reference Model [Evain98].

The Application Programming Interface (API) is crucial in the software architecture of digital television. Not only must the applications be downloaded in a standard way, but also the platform to run these applications must have a series of standardized software interfaces.

In fact, API is a built-in programmer's toolkit for requesting data objects or services resident on a particular operating system [Jacklin97]. It's an interface to an operating system. Using the APIs, a programmer writing an application can make requests to the operating system. There may exist multiple APIs depending on the system configuration. DVB Java platform includes Fundamental APIs, Presentation APIs, Data Access APIs, Service Information and Selection APIs, etc.

Each application that is developed will need to comply sufficiently with the reference model to ensure cross-platform interoperability. Openness, abstraction, evolution, and scalability are the main requirements of APIs.

2.2 Streamed Media API

Streamed Media API is defined as part of the DVB Java platform. The main purpose of Streamed Media API is *enhanced broadcasting*. Enhanced broadcasting means combining digital broadcast of audio/video services with downloaded applications which can enable local interactivity. It does not need an interaction channel. Streamed Media API allows Java applications to initiate data transfer and control the playback of time-based media. The Java Media Framework (JMF) [Sun98] defined by JavaSoft forms the main part of DVB Streamed Media API together with additional restrictions, features, and extensions.

The JMF is an API for incorporating time-based media into Java applications and applets to present media such as audio and video and to allow integration with the underlying platform's native environment and Java's core packages, such as java.awt. Media display encompasses local and network playback of media.

According to JMF Player specification, the JMF Player APIs support both pull data source and push data source [Sun98]. In Pull Data-Source mode, the client initiates the data transfer and controls the flow of data from pull data-sources. Established protocols for this type of data include Hypertext Transfer Protocol (HTTP) and FILE. In Push Data-Source mode, the server initiates the data transfer and controls the flow of data from a push data-source. Push data-sources include broadcast media, multicast media, and video-on-demand. For broadcast data, one protocol is the Real-time Transport Protocol (RTP).

JMF Player APIs are designed to support most standard media content types, including MPEG1, MPEG2, QuickTime, AVI, WAV, AU, and MIDI. The JMF technology allows the capability to plug new decoders into its framework [Sun98]. If a new media type comes along, a viewer's set-top-box could be upgraded via software downloaded.

2.3 Graphical User Interface (GUI) API

The GUI API, which is defined in DVB Java platform, includes functionality to draw graphics/widgets on the output device and to input events from the input devices. It is based on Java Abstract Window Toolkit (AWT) with additional TV extensions.

The java.awt package provides a user interface API to allow applications written in Java to generate graphical output and receive user input events. The AWT was designed to provide a common set of tools for GUI design that work on a variety of platforms. Fig. 2. illustrates the inheritance relationship among AWT widgets.

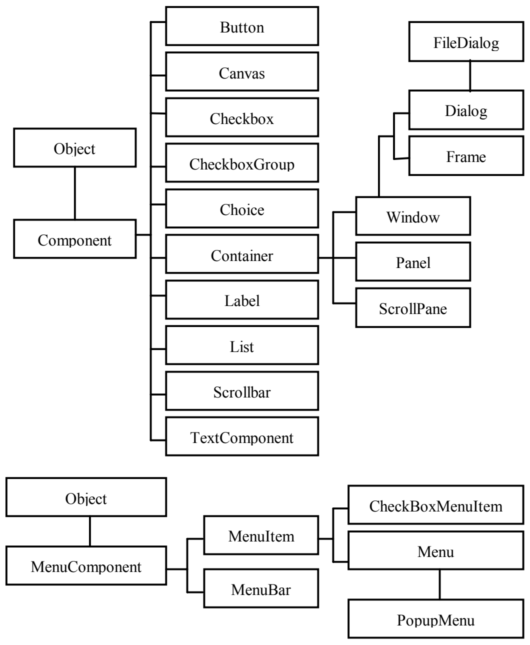


Fig. 2. Inheritance Relationship of AWT Widgets [Cornell99].

DVB will choose some of AWT components as a resident GUI widget set in the set-top boxes. The possible candidates would be Button, Checkbox, CheckboxGroup, Dialog, Label, Panel, TextArea, TextComponent, and TextField. The main reason for resident UI widgets in set-top boxes is to support limited bandwidth networks. This would allow applications to scale from limited bandwidth to higher bandwidth without extensive re-authoring.

The GUI elements provided by the AWT are implemented using each platform's native GUI toolkit, therefore preserving the “look and feel” of each platform. This is one of the AWT's strongest points. The disadvantage of such an approach is that a GUI designed on one TV set may look different when displayed on another.

The GUI components supplied by AWT are called heavyweight components [Cornell96]. A heavyweight component means that a native GUI component is used to display each Java component. This technique gives Java applications the same look and feel as other applications written for a particular platform. These components are considered heavy because they require twice as many classes to implement (i.e., a Java class plus its associated native class). They also have the unfortunate side effect of being opaque, which means that they can't be used to implement components with transparent regions, or components of non-rectangular shapes.

Java Swing is a new GUI component toolkit [Geary96]. It offers many more components, a common appearance, and identical behavior across platforms. One of the key factors contributing to Swing's importance is that each GUI component in the Swing set is a lightweight component. Lightweight components have no native twin. They are free to implement their own look and feel. However, Swing has a large code size that is the most important criteria of developing GUI in set-top boxes. Whether or not Swing will be selected for DVB GUI API is still an open issue.

3. IMPLEMENTATION

Based on the above technical background we introduced, in this chapter, we will present our work about Java user interface (i.e., the user interface of interactive television services or applications in set-top-boxes). The structure of Java user interface, how to control look and feel, overlay widgets on the video, and a case study will be discussed.

3.1 Structure of Java User Interface

Fig. 3. illustrates the basic structure of Java user interface. The Java user interface is composed of GUI and broadcasting content. The GUI includes graphics and user input as so called Look and Feel. Graphics means the visual presentation of widgets. Remote control, keyboard or virtual keyboard are needed for user input. Broadcasting content consists of video, audio, subtitles, teletext, and data. In a nutshell, Java user interface creates a visual presentation of the information by manipulating GUI widgets and video/audio.

Java user interface that we built is based on presentation APIs as well as fundamental Java APIs which are specified in the DVB Java Platform.

Presentation APIs include Graphical User Interface (GUI) API and Streamed Media API.

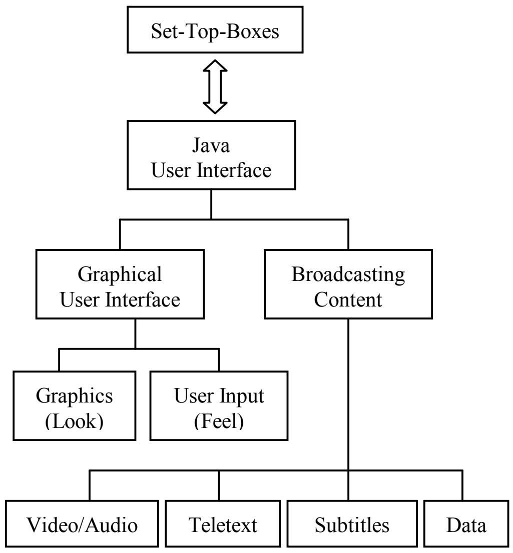


Fig. 3. Basic Structure of the User Interface in Digital TV.

3.2 Video

One of our goals of using Streamed Media API was to present the broadcasting content (e.g., video) coupled with GUI widgets on digital television screen.

The Player with pull data source was successfully created and controlled in our application. The Player was created with a *MediaLocator* [Gordon99] according to

MediaLocator mediaLocator = new
MediaLocator(URL);
Player player =
Manager.createPlayer(mediaLocator);

The *URL* is a source pointing to a DVB service (e.g., video). In our application, it is a compressed MPEG1 file. We haven't tested push data source because the final test platform is not ready yet. However, it should not affect the implementation of Java user interface. No matter what kind of data source is used, the compressed video stream is rendered in a Java AWT Component as discussed later.

A *Player* can be in one of six states (cf. Fig. 4): *Unrealized*, *Realizing*, *Realized*, *Prefetching*, and *Prefetched* and *Started* [Sun98]. The first five states correspond to a Stopped state. In normal operation, a Player steps through each state until it reaches the *Started* state.

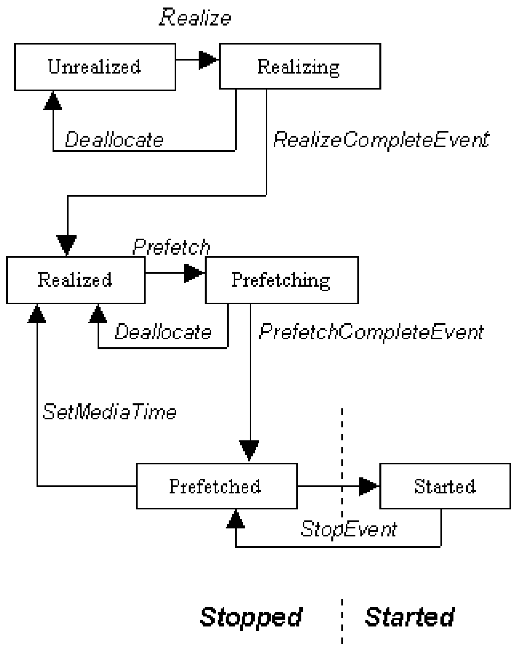


Fig. 4. State Diagram of JMF Player [Sun98].

When the method *player.realize()* is called, the player moves from the *Unrealized* state into the *Realizing* state. The *Realizing* Player is in the process of determining its resource requirements.

When the Player finishes *Realizing*, it moves into the *Realized* state automatically. When *player.prefetch()* method is called, the Player moves from the *Realized* state into the *Prefetching* state. When the Player finishes *Prefetching*, it moves into the *Prefetched* state automatically. A *Prefetched* Player is ready to be started.

At last the *player.start()* method puts the Player into the *Started* state. A *Started* Player's time-base time and media time are mapped and its clock is running, though the Player might be waiting for a particular time to begin presenting its media data.

The method *player.getVisualComponent()* returns an AWT Component that we added to our application window to render the DVB MPEG video stream.

Scaling video and positioning it on the screen in digital television are important requirements. We used *setBounds()* and *setLocation()* methods returned by *player.getVisualComponent()*.

3.3 Overlay of GUI Widgets on the Video with Transparency

In JMF 1.1 it is not possible to draw graphics/widgets or text over the video directly from the *VisualComponent* of the video. To circumvent this, we added GUI widgets to front of the components list and set layout manager to null. This way, one can draw whatever required in the widgets.

One of the key requirements in digital television environment is to overlay widgets, graphics and text over the video using transparency. This can be done in Java. However, JMF1.1 does not provide any lightweight component for rendering the video. The lightweight widgets are always covered by the heavyweight component of the video when the alpha channel is set. In JMF2.0 beta release, a *VideoRenderer* called *LightWeightRenderer* will be introduced. Thus, one can program the *PlugInManager* to use the *LightWeightRenderer* and create a lightweight component. We will use this method and place transparent widgets over the video.

There is an alternative way to place a transparent component over the video. Usually, each set-top-box has a graphics processor which supports colors with transparency [NorDig98]. This has been specified in GUI API in DVB Java platform as a TV extension.

3.4 Controlling Look and Feel

For the broadcasters it is very important that they can precisely control the look and feel of their applications and services. If they cannot control the look and feel of the resident widget set they won't use it.

As we know the AWT contains a Button class (cf. Fig. 2.), but Button is implemented through platform-dependent GUI components. Thus, one cannot draw into a button and expect the image to display properly. We replaced those heavyweight components with lightweight components by subclassing *Component*, *Container*, *Canvas*, or *Panel* depending on widgets we used. The widgets derived from Canvas or Panel are with rectangular regions, while those widgets derived from Component or Container can have any shapes. The images we used are single colored bitmap files. *TVButton* is frequently used in digital television environment. It can readily display an image with a label instead of just text label, and could also display border. The border can be shaded 3D-style border.

Usually, one can override *paint()* method inherited from Component class for drawing images or text to improve the performance and to eliminate flashing. *TVButton* flashing is not visible, but other widgets with bigger size and animation in a widget do flash. We used two techniques to eliminate flashing, one is overriding the *update()* method and another is implementing double buffering.

However, *Component*, *Panel*, *Canvas,* or *Panel* class doesn't provide any of the typical button-press effects. In digital television environment, it is very important to get user input and to have focus for navigation effects. The magic of handling user input and focus navigation was done by the event handler interface (i.e., derived from *KeyListener* interface) of *TVButton*. *TVButton* supports two states: focused and non_focused for navigation.

In digital television environment, the widget set must enable the use of traditional remote control and can not require a mouse and a computer style keyboard. At most a virtual keyboard can be used. In practice, we used keyboard codes to simulate remote control codes.

There are many possibilities to change or control the look and feel of the GUI widgets in applications. One way is to send bitmap images of all widgets to the TV viewer's set-top-boxes; Another way is based on lightweight components, every broadcaster can write his own widget set and will not use widgets located in set-top boxes; or replacing the look of the widgets of the resident widget set in set-top boxes.

3.5 Demonstration

We implemented an application user interface called *Screen Information Service* for ice hockey broadcasting program, which is a typical application for interactive television. Our experimental hardware platform was pentium II PC and software developing toolkits consisted of Java Development Kit (JDK1.2) and JMF1.1 together with our user interface class library.

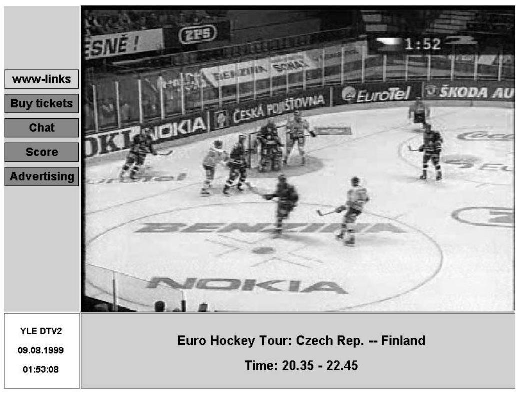
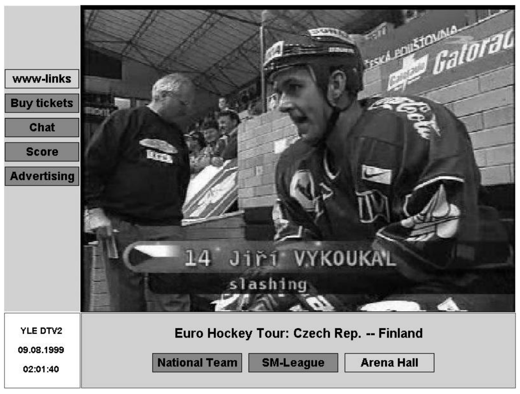


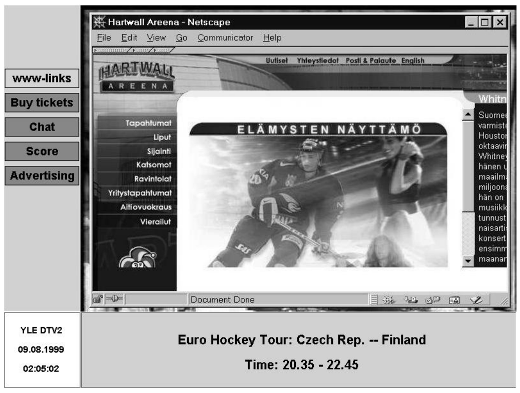
Fig. 5. The Main User Interface of Screen Information Service.

Fig. 5 shows the screen shot of the application's main page layout. It consists of four rectangular areas. The video is rendered on the top-right rectangle. The bottom-left rectangle area displays a timer and the channel logotype (YLE DTV2) of Finnish broadcasting company. The main menu is listed on the top-left rectangle area. The main menu is composed of five *TVbutton*s and each button indicates one function. Navigation of the functions is done by pressing up or down arrow of the remote control (i.e., in this case the keyboard). The information of user interaction is displayed on the bottom-right part which we call interaction area.

In the following, we briefly explain the functions of the application.



(a)



(b)

Fig. 6. The User Interface of *www-links*.

If the *www-links* button is selected, a popup menu is displayed in the interaction area (cf. Fig. 6(a)). It shows three buttons. The viewer can browse Finnish national team, SM-League, or Arena Ice Hockey Hall home page. If the viewer wants to browse Arena Ice Hockey Hall, then a browser is loaded and the www-page of Arena Ice Hockey Hall is displayed (cf. Fig. 6(b)).

When the *Buy tickets* button is selected, the name of the company (Lippuluukku) selling the tickets appears in the interaction area (cf. Fig. 7.). If the viewer selects the button, he or she can buy tickets from the company's web site for future games.

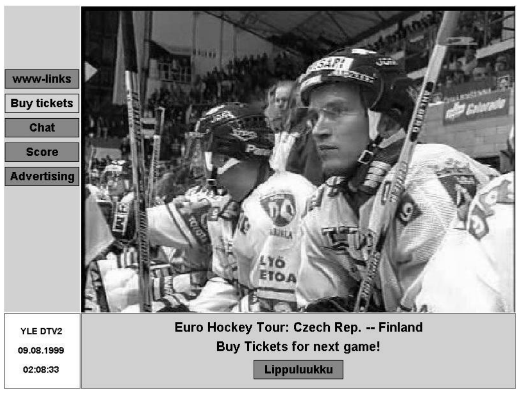
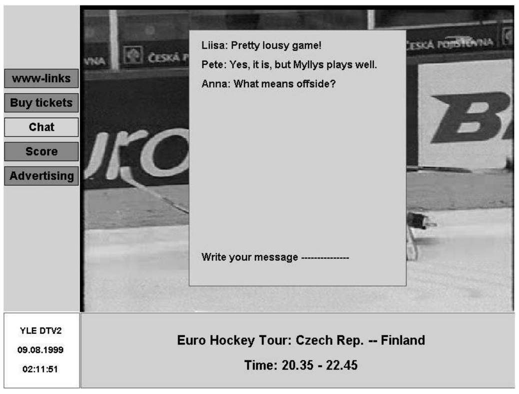


Fig. 7. The User Interface of *Buy tickets*.

If the *Chat* button is selected, the viewer is requested to give his or her name. After that, the viewers can chat during the game and give comments about the game (cf. Fig. 8.).


Liisa: Pretty lousy game!
Pete: Yes, it is, but Myllys plays well.
Anna: What means offside?

Write your message --------------

Fig. 8. The User Interface of *Chat*.

When the *Score* button is selected, the score of the game is scrolled up (cf. Fig. 9.).

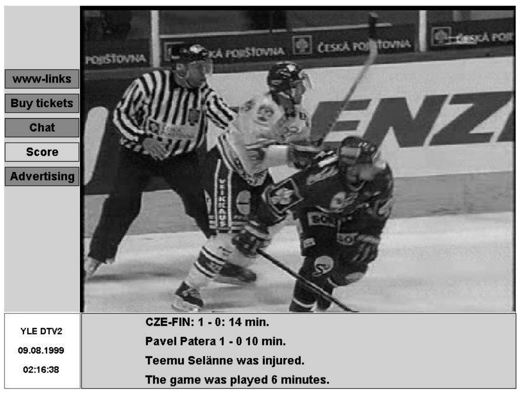
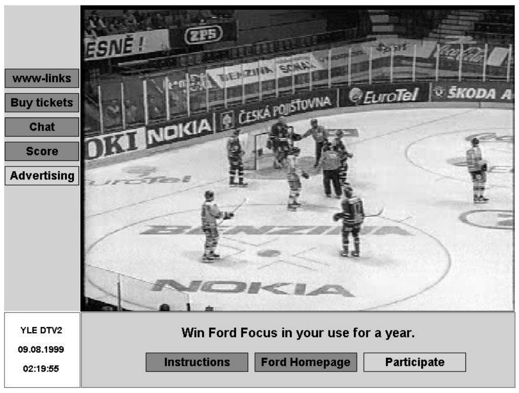

CZE-FIN: 1 - 0: 14 min.
Pavel Patera 1 - 0 10 min.
Teemu Selänne was injured.
The game was played 6 minutes.

Fig. 9. The User Interface of *Score*.

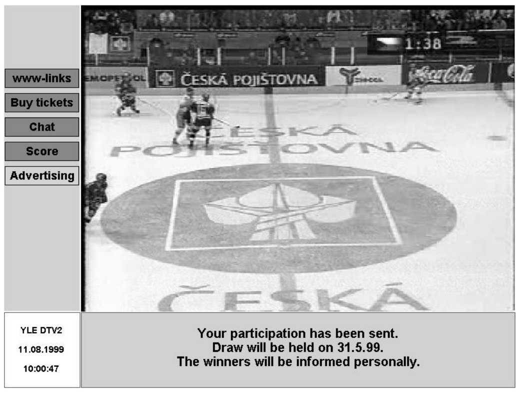
When the *Advertising* button is selected, a popup menu with three choice buttons is displayed in the interaction area (cf. Fig. 10(a)). The viewer can read instructions, browse the advertiser's homepage, or participates in the contest. Thus, the viewer has a chance to win a prize (cf. Fig. 10(b)).



YLE DTV2
09.08.1999
02:19:55

Win Ford Focus in your use for a year.

(a)



YLE DTV2
11.08.1999
10:00:47

Your participation has been sent.
Draw will be held on 31.5.99.
The winners will be informed personally.

(b)

Fig. 10. The User Interface of *Advertising*.

4. CONCLUSION

One of our goals in the Future TV project is to implement the user interface of digital television by using well-known APIs defined in DVB Java platform. So far, we have successfully developed a demonstration of user interface for interactive television service (i.e., interactive sports program). There remains a lot of implementation work to be done. We plan to embed our user interface library in set-top box for reuse of the resident functions; replace keyboard codes by remote control codes; implement a built-in navigator user interface including Electronic Program Guide; solve technical problems according to the requirements of APIs defined in DVB Java platform, etc.

Constraints of our system are set by the hardware and software architecture of the set-top boxes. Our biggest goal is to integrate the user interface library with real software system and make it work in real digital broadcasting environment.

ACKNOWLEDGEMENTS

We would like to thank Mr. Petri Koistila from the Future TV project for providing the example and translating the text of the figures (i.e., Fig. 5 – Fig. 10). The work is supported by the Future TV project which is funded by the National Technology Agency of Finland together with major Finnish television, telecommunications, and digital media companies.

The author would also like to acknowledge Nokia Oyj Foundation for the financial support.

REFERENCES

[Cornell96] Cornell,G, Horstmann,C,S, *Core Java™, SunSoft press*, 1996.
[Evain98] Evain,J,P, *The Multimedia Home Platform – an overview,* EBU Technical Review, Spring 1998.
[Fox98] Fox,B, *Digital TV comes down to earth*, *IEEE Spectrum*, October 1998.
[Geary99] Geary,D,M, *Graphic Java 2, Mastering the JFC: Swing, Sun Microsystems Press Java Series, Prentice Hall*, 1999.
[Gordon99] Gordon,R, Talley,S, *Essential JMF : Java Media Framework*, *Prentice Hall PTR*, 1999.
[Jacklin97] Jacklin,M, *The Multimedia Home Platform: on the critical path to convergence*, IBC Show Daily, September 1997.
[Luetteke98] Luetteke,G, *The DVB Multimedia Home Platform,* Philips Consumer Electronics, Hamburg, November 1998.
[NorDig98] NorDig I, *Digital Integrated Receiver Decoder Specification, for use in cable, satellite and terrestrial networks*, 1998.
[Sun98] Sun Microsystems, Inc., *Java™ Media Framework Programmer's Guide*, December 21, 1998.