CROSS-PLATFORM SMIL PLAYER

Kari Pihkala, Pablo Cesar, and Petri Vuorimaa
Telecommunications Software and Multimedia Laboratory
Helsinki University of Technology,
P.O.Box 5400, FI-02015 HUT
Finland

ABSTRACT

This paper presents the design and implementation of a portable SMIL player. The player has been written in Java and can be run on top of AWT, Swing, and *ftv* GUI frameworks. This allows running it in various platforms, e.g., PCs, PDAs, and digital television STBs. New media players can easily be added to the player, thus complying with the fundamental idea of SMIL, integrating various media formats. The performance of AWT, Swing, and *ftv* versions are evaluated to see how they fit to play SMIL. Finally, the requirements for the underlying graphical environment to play SMIL are given based on the evaluation.

KEY WORDS

SMIL, multimedia, Java, digital television, platform.

1. INTRODUCTION

The World Wide Web Consortium (W3C) is the organization specifying standards for the World Wide Web (WWW). In June 1998, Synchronized Multimedia Integration Language (SMIL) 1.0 Recommendation [1] was released as the multimedia format for the WWW. Three years later, in August 2001, a more profound SMIL 2.0 [2] was released. It had several additions, including animations, transition effects, and better support for user interaction. The SMIL 2.0 specification is split into modules, each defining a portion of the full SMIL 2.0 Language. Also, SMIL 2.0 Basic Profile is defined, consisting of ten basic modules for devices with limited capabilities. SMIL also supports content adaptation based on, e.g., the screen size, connection bit rate, selected language, and captioning, thus making it ideal for various devices.

Currently, the most used SMIL 2.0 player is the RealOne Player. Another famous player is Oratrix' Grins [3], which can also be used to compose SMIL presentations. Microsoft Internet Explorer 6.0 also supports HTML+SMIL [4], which is one proposed profile of SMIL. There are also several other SMIL players, some written in Java [5][6]. These use Java Media framework (JMF) [7] to play audio and video. One limitation of, at least some of the players, is that they are designed to run in certain platforms and not to be portable. Having a portable player has two advantages. First, multimedia can be represented in a wide family of devices. Second, using the same player will result the presentation to play consistently in all platforms.

This paper describes a design and implementation of a SMIL player, which can easily be ported to various platforms. The target platforms are a PC desktop, a Personal Digital Assistant (PDA) and a digital television set-top box (STB). These devices differ, primarily, in two aspects. The first one refers to the input and output devices. For example, a PC uses a mouse whereas a STB uses a remote control. The second one refers to the underlying hardware capabilities. For example, a PDA is a device with more memory restrictions than a PC. To implement a portable player, the decision was to use Java programming language. First, because it is a platform independent language. Second, because it is the "de facto" multimedia language standard, both Mobile Information Device Profile (MIDP) and Multimedia Home Platform (MHP) specifications use it.

Interactive applications need a Graphical User Interface (GUI) to communicate with the user. GUIs are built around a toolkit of widgets [8]. These widgets (e.g., Button, List) are ready-made screen items. Therefore, the developer does not have to deal with the underlying system. The core Java platform packages included a GUI framework, called Abstract Window Toolkit (AWT). Then, when a more powerful framework was needed, Sun developed Swing as an extension of AWT. Both AWT and Swing are tailored to the PC environment. For that reason, the digital television area needed a specific GUI framework. In all the European countries, the framework selected was Home Audio/Video interoperability (HAVi) Level 2 User-Interface [9]. Future TV (*ftv*) is our implementation of HAVi.

So far, the SMIL player runs on top of AWT, Swing, and *ftv* GUI frameworks. The player supports the SMIL 2.0 Basic Profile with additional BasicAnimation, BrushMedia, CustomTestAttributes, and EventTiming modules. The player has been developed to run as a part of an XML browser, X-Smiles [10], but can also be run alone. X-Smiles is a Java based XML browser intended for embedded devices, e.g., PDAs, mobile phones, and

STBs. Porting the SMIL player is the first step to reach this intention. The browser supports, e.g., SMIL, Scalable Vector Graphics (SVG), XForms, XML Events, Extensible Stylesheet Language Formatting Objects (XSL FO), and XHTML. The X-Smiles browser is available as open source at www.x-smiles.org.

This paper is structured as follows. In section 2, the features of platforms are studied. Section 3 describes the design and implementation of the SMIL player on top of the frameworks. Section 4 evaluates suitability of different GUI frameworks to display SMIL, while the last section draws conclusions and unveils the future.

2. PLATFORMS

This section describes the features of devices, introduces three GUI frameworks used in the paper, AWT, Swing and *ftv*, and finally discusses different alternatives for the graphic system architecture.

2.1 Device Features

Before designing the player, the different features of the devices were studied in order to understand the restrictions beforehand. These features were categorized into two groups: input and output features, and hardware restrictions. Input and output features include the screen size and resolution, the input device, and the viewing distance. Hardware restrictions include memory footprint, RAM memory, and CPU speed. Table 1 shows the input and output features. Table 2 shows the hardware restrictions.

Device	Screen Resolution (pixels)	Viewing Distance (feet)	Input Device	Screen Size (inches)
Digital Television	720x576 (minimal)	10-15	Remote Control	13 to wall size
Mobile Phone	84x48 to 120x130	Less than 1	Key Pad	Up to 6
PDA	160x160 to 240x320	Less than 1	Keyboard/ Stylus	6
PC Desktop	640x480 to 1800x1440	1 to 2	Mouse/ Keyboard	13 to 28

Table 1. Features of different devices [11].

Device	Storage Memory (Mb)	RAM memory (Mb)	CPU speed
Digital Television	16.000-40.000	At least 32	233-566 MHz
Mobile Phone	0.5	0.2	20 MHz
PDA	32	32-64	206 MHz
PC desktop	40.000	128-256	1.4 GHz

Table 2. Hardware restrictions of different devices.

Considering these tables, some basic restrictions were taken into account. Some of the are the following:

- The size of the fonts in digital television should be bigger than in other devices, since the viewing distance is longer.
- Remote control events should be included in digital television, since mouse is not considered as an input method.
- Several resolutions should be provided to make applications run on different platforms.
- Memory is restricted in digital television and mobile phone environments, so a small framework should be provided.

2.2 Java GUI Frameworks

AWT is a toolkit to build GUIs for Java software. AWT divides the task of supplying GUI services between the high-level GUI classes and the platform-level peer components supplied by the underlying platform. Java software interacts only with the high-level GUI classes, while AWT maps these to peer components. Since AWT needs a native toolkit, it is called peer or heavyweight framework. On the other hand, Swing [12] does not make use of native toolkit components, except at the container level. It has no native resources of its own, that is why it is called peerless or lightweight framework. In order to be painted, each component overrides the paint(g) method.

HAVi is a "TV-friendly" GUI framework. It uses java.awt package as a core. Although most of AWT classes are not HAVi compliant, some of them have been included in the specification, e.g., java.awt.Color [9]. HAVi framework is a pure Java lightweight one. It defines a set of widgets, in which the look and feel is separated. In addition, it specifies new events, typical for digital television environment, such as remote control events. *ftv* was implemented at the TML lab, Helsinki University of Technology. It is a Java GUI framework based on HAVi specifications [13].

ftv is composed of text, animation, text input, and graphic widgets. These widgets can be visible (e.g., ftvStaticText), navigable (e.g., ftvText), actionable (e.g., ftvTextbutton) and switchable (e.g., ftvToggleButton). In order to separate the look and the feel, each widget has an associate look class. This class takes care of rendering. When a widget is created, a default look class is used. Also, the developer can implement specific looks depending on the needs. Figure 1 depicts the basic *ftv* widgets. Finally, *ftv* is portable, so it can run over any implementation of AWT, e.g., Sun's AWT, or Kaffe AWT.

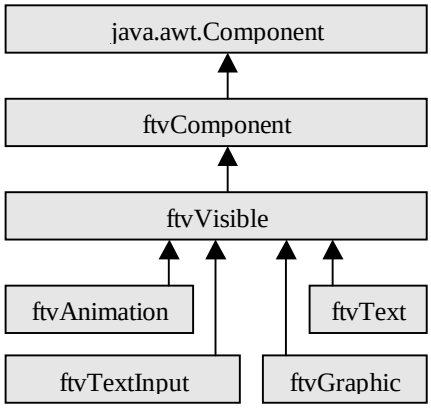

Figure 1. UML diagram of basic *ftv* widgets.

2.3 Graphic Systems Alternatives

The graphic system software is a very important topic, since it determines the amount of memory required in the system, as well as the capabilities of the framework. Figure 2 depicts three different architectures for the graphic system software. The first one is a heavyweight approach, e.g., AWT, in which a native toolkit and a native windowing system are needed. The second one is a lightweight approach, in which the toolkit is implemented only at the Java level, but a windowing system is needed, e.g., Swing. The third one only uses a graphic library, which it is able to paint directly in the framebuffer device, e.g., Kaffe.

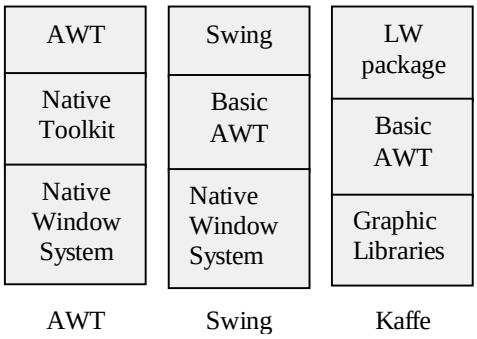

Figure 2. Graphic system [14].

The first architecture, since it is a heavyweight one, uses twice the number of widgets than the rest, i.e., it has Java and native widgets. In addition, the look of the applications varies depending on the underlying platform. Also, the number of native calls is per component and not per container as in a lightweight approach. The second and third architecture use a native window system, e.g., X-Windows in Linux, which takes a lot of memory. At the same time, the third option provides less functionality than the rest, because of the lack of a native window system already implemented.

As a conclusion, the better option to take will depend on the hardware capabilities of the system. The first option is followed nowadays in the MIDP specification. Swing is used in PC desktop environments, since the memory footprint and RAM memory are not restricted, and provides a full GUI functionality. In the case of a digital television, the architecture will depend on the underlying hardware. In the case of no memory restrictions, the second option can be taken, with the use of HAVi instead of Swing. In a more restricted environment, the third option will be more suitable.

3. DESIGN AND IMPLEMENTATION

This section describes the design of the SMIL player, so it can run in various GUI frameworks. Running it in various GUI frameworks enables running it in various devices. Figure 3 depicts the design of the player to achieve this goal.

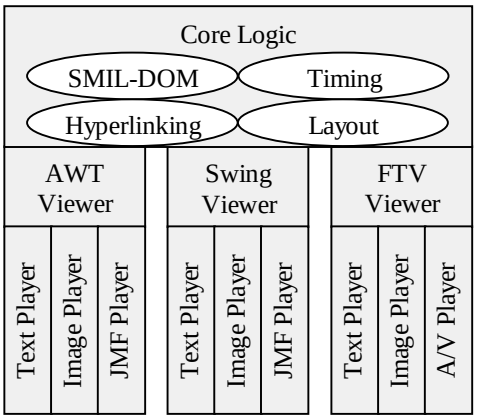

Figure 3. Design of the SMIL player.

The SMIL player is divided into three layers. The Core Logic is the heart of the SMIL player. It handles timing, decides the layout of the presentation, resolves hyperlinks, etc. When a SMIL document is opened, it is parsed with an XML parser, which constructs a Document Object Model tree. Each element in the tree knows its own behavior, taking care of timing and layout. This results in a robust and dynamic internal structure for the presentation [15]. The Core Logic is completely GUI independent, making it possible to run it in any Java environment with any GUI framework.

The Core Logic layer utilizes the Viewer layer to draw media on the screen according to the timing information. Various devices, e.g., desktop computer, digital televisions, and mobile phones use different kind of GUI frameworks. A viewer can be implemented for any possible GUI framework. So far, a viewer has been created for Swing, AWT, and *ftv*. The Viewer layer also provides a user interface for the Core Logic, allowing opening, playing, and stopping presentations. This layer is

affected the most by the device features discussed in section 2. Figure 4 depicts a screenshot of the digital television version, showing the user interface widgets at the top. Larger fonts have been used and the user can move the focus with keys.

Figure 4. Screenshot of the SMIL Player.

On bottom of the Figure 3 are the media players, which are viewer and GUI framework specific. Because the interface between the viewer and media players is well defined, it is easy to add new media players to viewers. Currently, text and image media formats are supported by custom players. In order to play audio and video, JMF is used. JMF is capable of playing several media formats, e.g., MPEG-1, AVI, WAV, and MP3. It could have been extended to play also text and images, but the performance would have been worse than with the custom players [16]. Having JMF only as a continuous media player, the SMIL player can be run even if JMF is not installed. JMF cannot be run in the framebuffer version of *ftv*, thus custom audio and video players are needed.

When running the SMIL player in X-Smiles, a viewer based on the Swing version is run. The X-Smiles browser user interface controls the SMIL player. Also, an additional XML media player is available, enabling displaying SVG, XSL FO, or XHTML content in SMIL presentations.

4. EVALUATION

In this section, the performance results of the SMIL player running on the three GUI frameworks are given, to see the spectrum of devices it can be run on. Memory usage and time to parse and display the document depicted in Figure 4 was measured. In addition, the memory footprint of the Java classes is provided. Finally, the basic requirements of a GUI framework to run a SMIL player are given.

4.1 Time and Memory Usage

Table 3 shows the results running the SMIL player on the different GUI frameworks. In each case, the operating system (Linux) and the Java Virtual Machine (Kaffe) were the same to have comparable results. Kaffe had a graphic system architecture painting directly in the framebuffer and a Just-In-Time (JIT) compiler to speed it up.

	AWT	Swing	*ftv*
Memory (kb)	11.534	14.680	10.486
Time (ms)	2.013	3.250	1.627

Table 3. The time and memory usage for the SMIL player.

ftv package is the fastest and less memory consuming. This is because it is a special purpose digital television package. Having a specific purpose, the number of classes is lower than in the other packages. In the case of AWT, the main difference is the use of a native toolkit, i.e., more classes. Swing is a general-purpose framework. Hence, it provides more functionality, but at the same time needs more memory. As a result, it can be noted that the SMIL player can be run in restricted devices, such as PDAs with *ftv* or AWT. It cannot be run in mobile phones, due to lack of JIT compiler.

4.2 Memory Footprint

It was important, as well, when talking about restricted devices, to study the memory footprint of the different alternative GUI frameworks. Table 4 depicts the memory needed for the packages in use.

Package	Memory (kb)
Kaffe APIs	1.075
swing.jar	2.420
ftv.jar	96
xerces.jar	862
smil.jar	250

Table 4. Memory needed for the packages.

The first conclusion, very similar to the previous results, is that Swing is a huge package, but at the same time very powerful. Hence, it would not be an option for restricted devices. *ftv*, on the other hand, provides the required functionality and is small. Finally, for restricted devices, the XML parser memory footprint should be minimized. Again, the results show that it is possible to run the SMIL player in restricted devices, such as PDAs, but not in mobile phones.

4.3 Requirements

The following points were discovered during the development of the SMIL players. These features should be available in the GUI framework in order to implement a SMIL player.

- **Double-buffering.** The GUI framework should have double-buffering or a light rendering system to prevent animations from flickering. AWT uses heavyweight components, causing flickering. On the other hand, *ftv* was implemented using double-buffering.
- **Z-Indexing.** It should be possible to control the order of various media components, to support z-indexing. Again, AWT does not have a good support for this. *ftv* does not provide, either, a good support for z-indexing, yet.
- **Transparency.** The GUI framework should support transparent backgrounds, for instance, in texts. This is possible in AWT. Also, *ftv* allows transparency as opaque or transparent look. However, an alpha channel should be included in the future, for better control over transparency.
- **Mouse/Focus system.** A way to activate links is required. In AWT and Swing, mouse clicking is possible. In *ftv*, mouse is not available, thus a focusing system is required. It includes a new focus system, in which the focus is transferred from the current widget to another specific one.
- **Event system.** The SMIL EventTiming module allows listening to mouse and other user interaction events. In *ftv*, this can be problematic, if the user is able to focus only on the links. However, allowing focusing on all media components will result in too many focus points, and chaos. The decision has been to design SMIL presentations with a careful use of EventTiming.
- **Media players.** The framework should be able to display at least the basic media formats, such as text with various fonts, JPEG images, and perhaps audio and video. In *ftv*, custom players for audio and video have to be developed.
- **Video capabilities.** The framework should allow mixing video with other media without inconsistencies. Using JMF, video may be presented in a heavyweight component, resulting in a loss of control over z-indexing in Swing.

In addition to the above points, it is not possible to implement some of the SMIL modules, such as transition effects. Transition effects module requires a control over the alpha channel of the media with a fast image processor, which are features not available in the current GUI frameworks. Also, SMIL MultiWindowLayout Module provides support for opening new windows; therefore platforms without windowing system require attention.

4.4 Devices

The SMIL player presented in this paper has been tested in different platforms. It worked in a PC desktop, both Linux (X-Windows and Framebuffer options) and Windows Operating Systems. Also, it was tested in the digital television STB prototype introduced by Sivaramana et al. [17], and finally, in Compaq iPaq PDA.

5. CONCLUSION

Separating the SMIL synchronization logic from the user interface and media players makes porting a SMIL player easy. A SMIL player has been created for AWT, Swing, and *ftv*. Also, having a clear interface for individual media players makes it easy to add support for new media formats. This is in line with the fundamental idea of SMIL, integration of media. In the future, new modules, e.g., MultiArcTiming and ExclTimeContainers, will be added to the SMIL Core Logic, thus making them available for all platforms.

The designed SMIL player runs in PC desktop (both Linux and Windows OS), IPAQ PDA, and a STB prototype. The GUI framework to be used depends mostly in the restrictions of the platform. The underlying GUI framework should have capabilities for double-buffering, z-indexing, transparency, link focusing, and various media players. AWT does not have these. If the platform is a PC desktop, the best option is the use of Swing, since it provides support for all mentioned capabilities. In more restricted environments, such as digital televisions or PDAs, the solution of using *ftv* over graphic libraries, painting directly in the framebuffer, seems more suitable. *ftv* has still its limitations, which will be considered in the future.

Various applications of SMIL include slide shows, advertisements, interactive guides, games, and education programs. Adaptivity of SMIL gives opportunity to customize the applications for various target devices. The good results obtained porting the SMIL player will encourage us to start porting the whole X-Smiles browser for various target platforms.

6. ACKNOWLEDGEMENTS

The authors would like to thank the following people. Mr. Jukka Santala for providing a stable and reliable Kaffe VM and PDA environment. M.Sc. Ganesh Sivaraman for starting to use the Kaffe+framebuffer idea in the project. The author Kari Pihkala would also like to thank Nokia Oyj Foundation for providing a scholarship.

REFERENCES

[1] P. Hoschka et al., Synchronized multimedia integration language (SMIL) 1.0 specification, *W3C Recommendation*, June 15, 1998.

[2] L. Rutledge, SMIL 2.0: XML for Web multimedia, *IEEE Internet Computing, 5*(5), 2001, pp. 78-84.

[3] D. Bulterman, et al., GRiNS: an authoring environment for web multimedia, *World Conference on Educational Multimedia, Hypermedia and Educational Telecommunications, ED-MEDIA 99*, Seattle, WA, USA, 1999.

[4] D. Newman et al., XHTML+SMIL Profile, *W3C Note*, Jan. 31, 2002.

[5] J. Xia et al., Design and Implementation of a SMIL Player, *Proceedings of SPIE, 3648*, San Jose, CA, USA, 1999, pp. 382-389.

[6] P. N. M. Sampaio, C. A. S. Santos and J. P. Courtias, About the semantic verification of SMIL documents, *International Conference on Multimedia and Expo*, New York, NY, USA, 2000, pp. 1675-1678.

[7] R. Gordon and S. Talley, *Essential JMF: Java Media Framework*, (Upper Saddle River, NJ: Prentice Hall, 1998).

[8] D. R. Olsen, *Developing user interfaces* (San Francisco, CA, USA: Morgan Kaufmann Publishers, 1998).

[9] HAVi Organisation, HAVi level 2 User Interface, 2001, available at www.havi.org

[10] P. Vuorimaa et al., A Java based XML browser for consumer devices, *the 17th ACM Symposium on Applied Computing*, Madrid, Spain, March 10-13, 2002, pp. 1094-1099.

[11] Sun Microsystems, Truffle graphic customization guide, White paper, *referred on 11th June 2002*, available at *http://java.sun.com/products/personaljava/truffle*

[12] R. Eckstein, M. Loy and D. Wood, *Java™ swing* (Sebastobol, CA, USA: O'Reilly & Associates, 1998).

[13] P. Cesar and P. Vuorimaa, A graphical user interface framework for digital television, *Proceedings of the 10th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision, Posters, WSCG'2002*, Plzen, Czech Republic, 2002, pp. 1-4.

[14] G. Sivaraman and P. Vuorimaa, Compact windowing system for mobile devices, *Proceedings of the 2nd International Symposium on Mobile Multimedia Systems & Applications, MMSA2000*, Delft, The Netherlands, 2000, pp. 134-141.

[15] K. Pihkala and P. Vuorimaa, Dynamic SMIL Player, *International Conference on Multimedia and Expo*, Lausanne, Switzerland, 2002.

[16] K. Pihkala, N. von Knorring, and P. Vuorimaa, SMIL in X-Smiles, *Proc. the Int. Conf. on Distributed Multimedia Systems*, Taipei, Taiwan, Sept. 26-28, 2001, pp. 416-422.

[17] G. Sivaraman, P. Cesar, and P. Vuorimaa, System Software for Digital Television Applications, *Proc. 2001 IEEE Int. Conf. On Multimedia and Expo, ICME2001*, Tokyo, Japan, August 22-25, 2001, pp. 784-787.