SMIL IN X-SMILES

Kari Pihkala, Niklas von Knorring, and Petri Vuorimaa
Telecommunications Software and Multimedia Laboratory,
Department of Computer Science and Engineering,
Helsinki University of Technology,
P.O. Box 5400, FI-02015 HUT, Finland.
Email: kari.pihkala@hut.fi, niklas.knorring@mindonmove.com, and petri.vuorimaa@hut.fi

ABSTRACT

World Wide Web Consortium has specified Synchronized Multimedia Language (SMIL), which is intended to bring multimedia into World Wide Web. The most popular browsers still don't support SMIL in its full, which slows down the utilization of SMIL. In this paper, we describe our implementation of a SMIL 1.0 player. The player is part of our X-Smiles browser, which is an open source XML browser.

Keywords: SMIL, multimedia, Java, browser, World Wide Web.

1 INTRODUCTION

Although World Wide Web (WWW) is originally a hypermedia system, it is developing towards a multimedia system [1]. One of the main reasons for this is that the WWW is increasingly used in new devices like digital television Set-Top Boxes (STB), Personal Digital Assistants (PDA), and mobile phones. These devices utilize multimedia in one form or another. As the new devices become more popular than personal computers as WWW clients, the demand for multimedia in WWW is increasing.

So far, the multimedia support in WWW has been limited. The most common solution to use continuous media in WWW browsers is to utilize so called plug-ins. These are special software components that can play some specific media type. The plug-ins are usually installed separately by the user to the browser, but they can also be pre-installed. The main disadvantage with plug-ins is that different media elements have no synchronization between them. Thus, multimedia is more like decoration than integral part of the content.

Synchronized Multimedia Integration Language (SMIL) [2] is a special effort by the World Wide Web Consortium (W3C) to make multimedia more integral part of the WWW. The SMIL is an Extensible Markup Language (XML) [3] based language, which can be used to define the relationships of different media elements in WWW applications. The SMIL allows, for example, the exact synchronization of different media elements in temporal dimension.

The original SMIL 1.0 recommendation [4] was released in June 1998. After that, a second Synchronized Multimedia Working Group (SYMM) was established in February 1999. The public working draft of the next version of SMIL 2.0 [5] was released in March 2001. The main difference between these two versions is that SMIL 2.0 is far more extensive than SMIL 1.0. The SMIL 2.0 contains many new features, which have been grouped as modules (e.g., animation, content control, layout, linking, metainformation, timing and synchronization, transition effects, etc.).

The SMIL is also linked to other W3C activities. For example, the SMIL 2.0 animation module can be used together with Scalable Vector Graphics (SVG) [6]. In addition, the XForms [7] specification can be used to input information in SMIL applications. The SMIL has also connections to Motion Picture Expert Group (MPEG) activities. SMIL can be used to define content for MPEG-4 [8] applications, while MPEG-7 [9] can be used to define metadata for SMIL.

Although SMIL is gaining popularity, its use is still limited. The main reason for this is that only few editors and browsers are available. Perhaps, the best known SMIL tool is the RealSystems G2 Player, which utilizes SMIL for complex presentation composition. Other Java based experimental players are also available. The best known SMIL editor is the GRiNS editor [10], which can be used to create SMIL applications, which are compatible with both the official standard and RealSystems G2 Player. So far, none of the popular WWW browsers supports SMIL. Microsoft Explorer 5.5 supports some parts of SMIL 2.0, though.

In this paper, we present a SMIL 1.0 browser implementation. The X-Smiles browser [11] is a Java based browser, which supports, e.g., SVG, XForms, and Extensible Stylesheet Language Formatting Objects (XSL FO) [12] in addition to SMIL. Thus, it's an ideal environment to experiment with SMIL and related XML specifications. The X-Smiles browser is also available as open source at www.x-smiles.org.

The paper is organized as follows. In the following Section 2, we present the X-Smiles browser. After that, the SMIL implementation of X-Smiles is described in Section 3. Next, we discuss different performance issues related to the SMIL implementation. Finally, conclusions are given in Section 5.

2 THE X-SMILES XML BROWSER

This section describes briefly the X-Smiles browser and its components. The SMIL player has been implemented as one component of the browser.

The X-Smiles is a Java based XML browser intended for embedded devices, e.g., PDAs, mobile phones, and digital television STBs. The idea is that the next generation multimedia devices will have a small operating system (e.g., mobile Linux) and a Java virtual machine. The X-Smiles browser runs directly on top of these. All the services are developed using XML and related specifications, and they are displayed by the X-Smiles browser.

The architecture of the X-Smiles browser is depicted in Fig. 1. The architecture is composed of five major layers (from bottom to the top): XML processing, Browser Core Functionality, Markup Language Functional Components (MLFCs) & ECMAScript Interpreter, and Graphical User Interfaces (GUIs). Each layer is described in the following.

The XML processing module contains the XML Parser and XSL Processor. The XML Parser parses the XML documents. The XSL Processor processes the document according to the related XSL stylesheet, if applicable. The output is stored into a data structure, which can be accessed via the Document Object Model (DOM) Interface.

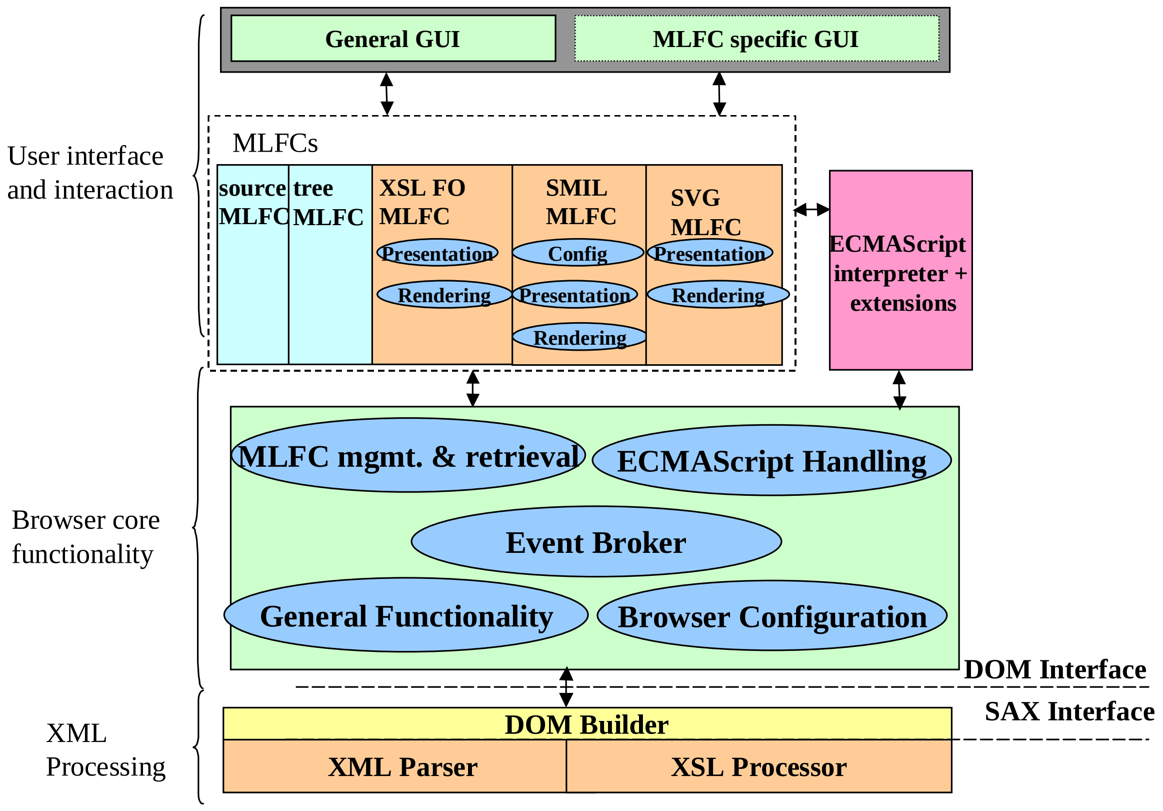


Figure 1. The internal architecture of the X-Smiles XML browser.

After the DOM tree has been constructed, it has to be rendered. Since there are different XML languages (e.g., MathML, ChemicalML, SMIL, XSL FO, and SVG) available, different rendering modules have to be used. In X-Smiles, this is achieved by using different MLFCs. Each MLFC knows how to render one specific type of XML language. The different MLFCs can be used at the same time. For example, an XSL FO document can contain a SMIL document. So far, we have implemented an MLFC for SMIL, XSL FO, and SVG.

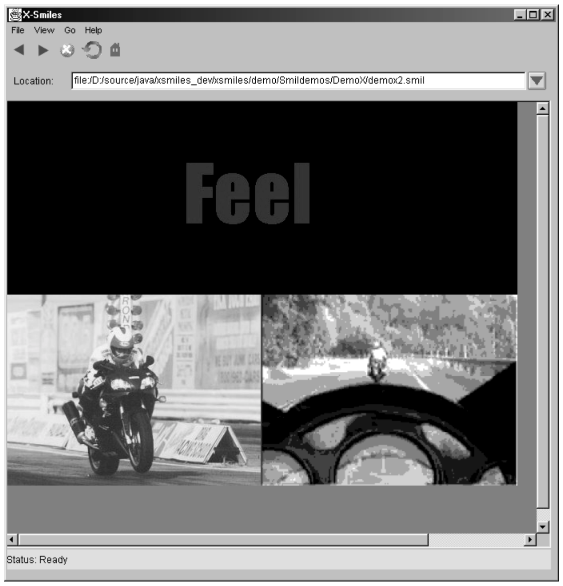
The core of the browser ties the different modules together. The main parts of the core are MLFC Management and Retrieval, ECMAScript Handler, Event Broker, General Functionality, and Browser Configuration. The MLFC Management and Retrieval unit takes care of loading the appropriate MLFC, which can also be retrieved over the network. The Event Broker forwards the events to other units. The ECMAScript Handler co-ordinates the operation of the ECMAScript interpreter. The General Functionality unit controls the GUI, document history, etc. Finally, the Browser Configuration unit is responsible for management of the different features of the browser such as the parser and stylesheet processor, home page, etc. For SMIL, the Browser Configuration is used to save user preferences, such as network bandwidth, language, and caption preferences.

The ECMAScript Interpreter runs the scripts contained in the XSL stylesheet. ECMAScript provides service developers a convenient method to implement interactivity. In SMIL, ECMAScript can be used, for example, to change the source URI of the image. This allows changing the image, if it is being clicked, or the mouse pointer is moved over it.

The original X-Smiles GUI was developed for desktop PCs. We have also developed a special user interface for digital television and are working on mobile versions of the GUI. The different GUIs are interchangeable, and the user can switch them even on the fly. Every GUI can have its own screen size. SMIL can be played on different GUIs and the resolution of the presentation can also be changed using the SMIL switch element.

3 IMPLEMENTATION OF THE SMIL MLFC

This section describes the main technical implementation issues of the SMIL MLFC used in the X-Smiles browser. The MLFC plays SMIL presentations, as depicted in Fig. 2.



Location: file:/D:/source/java/xsmiles_dev/xsmiles/demo/Smildemos/DemoX/demox2.smil

Status: Ready

Figure 2. SMIL presentation in X-Smiles.

The SMIL MLFC is able to play text, images, audio and video. This has been accomplished using the Java Media Framework (JMF). JMF is a package, which enables Java applications to play multimedia, such as audio and video. It is an extension to the Java 2 Platform giving developers an easy way to play multimedia platform independently.

Implementation of the SMIL MLFC raised the following technical issues:

1. Having an abstract representation of the source document in the form of a DOM tree, how should one convert this representation into a run-time SMIL presentation?

2. JMF supports only streaming media, such as video and audio. How should JMF be extended to support static media types such as images and text?

3. How should timing and synchronization issues be solved?

4. How the SMIL MLFC can interact with other MLFCs?

These issues are discussed in the following subsections.

3.1 Transformation from DOM to runtime objects

This subsection describes how the builder pattern [13] is used to transform a DOM representation of the SMIL

source XML code into a runtime representation of the SMIL presentation.
The SMIL MLFC, like all the other MLFCs in the X-Smiles browser, render the contents of an XML document by transforming the DOM representation of the XML document into a MLFC specific representation of that document. The SMIL MLFC specific representation of the document is a rendered SMIL document, that displays and synchronizes media elements according to the SMIL specification [4]. Fig. 3. gives an overview of the functionality of the SMIL MLFC.

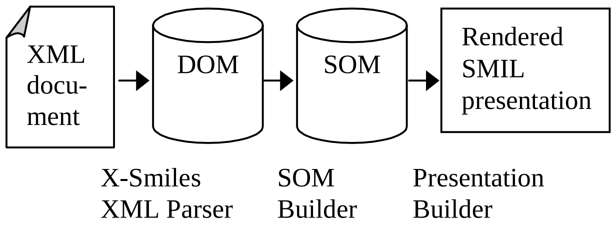


Figure 3. Transforming the DOM into a SMIL runtime representation.

The SMIL MLFC begins by transforming the DOM tree into a SMIL Object Model (SOM) using the SOMBuilder. The SOM is a collection of Java classes, each class representing a different element or construct in the SMIL language. The SOM can be thought of as a SMIL specific DOM. It can be used only with SMIL documents, but when used with SMIL documents, it provides a more natural interface to the elements and constructs of the document than DOM. The purpose of the SOM is to

1. Provide an easy to use interface to access the different parts of the SMIL document and

2. Perform validity checking on the SMIL document.

The SOM is then used as the starting point for the next transformation, which is the transformation from the SOM into the actual runtime SMIL presentation. This transformation is performed by the Presentation Builder. The Presentation Builder works by iterating through the SOM, creating players for each media-element and synchronizing elements that it encounters.

In order to create the players, the media for the players has to be fetched. Images and text are always fetched in advance into system memory, whereas video and audio can either be streamed using Real-Time Streaming Protocol (RTSP) or fetched in advance into system memory. The content type of the URI referring to the media file determines the type of player needed for the media. When all players have fetched their respective media files and are ready to be played, the presentation is started.

3.2 Extending the Java Media Framework

JMF is designed for streaming media, and the media types that it supports by default are either audio or video clips. The SMIL specification requires that a SMIL player must support static media, such as images, text, and text streams as well. Implementing support for static media types in a SMIL player that is based on JMF, such as the SMIL MLFC in X-Smiles can be done either by extending the JMF to support the static media types, or by implementing the players for the static media types without using JMF for timing and synchronizing.

JMF can be extended to handle custom media types, such as static images, text, and text streams. This has been done by implementing an AbstractPlayer class. The AbstractPlayer class was originally designed for this purpose [14]. The new players also have to be registered with PacketManager class. After registration, JMF can view images and text using the same interface as for audio and video.

Implementing players for static media types from scratch requires implementing similar timing and synchronization capabilities as those found in JMF. In X-Smiles, a class called MediaProxy encapsulates the JMF functionality as shown in Fig. 4. This class has five primary methods: init, prefetch, start, stop, and close. These methods handle all the JMF related processing. It is also possible to add a listener for this class. This way it is possible to get events about the end of media. It was fairly straightforward to rewrite this class to retrieve the images using Swing and to implement timing with a Timer class. The same was done for text media. The MediaProxy class was left to handle audio and video. A comparison between the performance of these two implementations is given in Section 4. These classes are called proxies, because they also cache data. If the same URL is encountered again in the same presentation, then the content of the URL is not retrieved again, but only reused from the cache memory.

3.3 Synchronization

A SMIL presentation consist of media elements that can be synchronized in various ways. The presentation thus has a temporal dimension, unlike, e.g., HTML documents. This temporal dimension can be represented by a time-line, where each media element starts at a given time and ends at another.

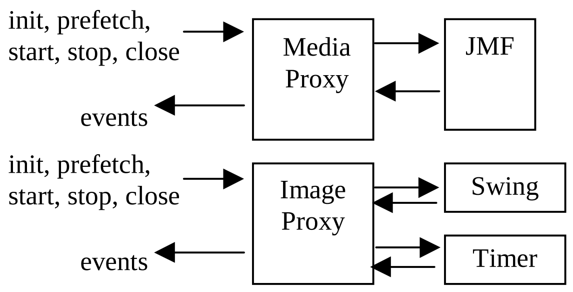

init, prefetch, start, stop, close
events
Media Proxy
JMF
init, prefetch, start, stop, close
events
Image Proxy
Swing
Timer

Figure 4. Implementation of JMF proxy and image proxy.

There were two alternatives in implementing the timing and synchronization of media elements in the X-Smiles browser. The first was to use the synchronization capabilities of the JMF to orchestrate the entire presentation. JMF players have advanced synchronization capabilities. For example, a player can be instructed to stop or start at the same time that another player stops or starts, or any number of seconds after the other player stops or starts. This makes it possible to express all the synchronization demands of SMIL using the timing attributes of the JMF players. The second alternative is to use the JMF players only to play the media, while synchronization is handled by proprietary scheduling. This involves creating a time line of the presentation and having a scheduler interpret this time line, starting and stopping media elements at times defined in the SMIL presentation.

After some consideration the first alternative was chosen, because it was much more straightforward to implement. There are downsides to this solution, most notably the fact that there exists no implicit time line of the presentation. Because of this fast forwarding or rewinding to arbitrary positions of the presentation (and not just to the beginning or end of a media element) is not possible.

3.4 Interacting with other MLFCs

The power of the X-Smiles browser is in its support for several XML standards. This allows mixing different XML languages. There are two ways to include another XML language into a SMIL presentation. The simplest way is to point the source attribute of a media object element to an XML file. This is possible, because the type of the media is not determined from the name of the media object element (e.g., text or audio), but the content type of the URI.

With SVG this works as follows. After determining the content type of the URI, the SMIL MLFC asks the browser to give it a component, which contains the SVG graphics. The browser will launch the SVG MLFC, which will handle the URI and returns a component for the SMIL MLFC. The returned component contains the SVG graphics. The SMIL MLFC will then show the component as it would show an image. Therefore, attributes such as begin and end can still be used to control the timing of the media object element.

The second approach is to include another XML language in the SMIL presentation using a different namespace. The two languages will then be mixed in the same file. In this case, the other XML language will be fed to another MLFC as it is read by the SMIL MLFC. We have used such an implementation to include XForms in SMIL. Every time an element in XForms namespace is encountered, it is fed to the XForms MLFC. This MLFC returns a component, which is then displayed by the SMIL MLFC. The XML standard allows mixing different namespaces in one element. Therefore, the XForms element can have attributes such as begin and end to specify the timing for SMIL. It can also have XForms attributes to control the XForms behaviour.

4 PERFORMANCE

Time to prefetch data and memory usage was measured for two versions of SMIL MLFC. One rendered presentations completely with the JMF, and one used Swing to render texts and images. The measurements were made with presentations, which had 1, 5, 10, 15, and 20 media elements. Data for the presentation was loaded from a local disk, so there was no slowdown caused by a network. We used the Performance Pack for Windows of the JMF 2.1 and Sun's Standard Edition of the Java Runtime Environment Version 1.2.2 with a JIT compiler. In the following, the differences of the two implementations are described. Only the performance for text and image components was evaluated, because only these are rendered differently in the implementations.

4.1 Text component performance

The simplest media format in SMIL is text format. It doesn't require a lot of memory or rendering power. Therefore, it is easy to measure and compare the two different SMIL MLFC implementations with text components. Five different presentations were used in the measurements. Each of them had different number of media elements as described earlier. Each media element referred to a small, ten character text file.

Fig. 5. and Fig. 6. show the results. Time keeping was started, when the SMIL MLFC was initialized, and stopped, when the SMIL presentation had been prefetched. Thus, the playback of the SMIL presentation was not included in the measurements. The playback time is irrelevant, because it is defined purely by the timing attributes in the presentation, which doesn't affect the performance. Also, the memory usage of the MLFC was measured, from the initialization of the SMIL MLFC to the end of prefetching. The Java garbage collector was run before initialization to get consistent result. In some cases, garbage collection took considerably long time (up to half a minute), and it was not reasonable to include it in the measurements.

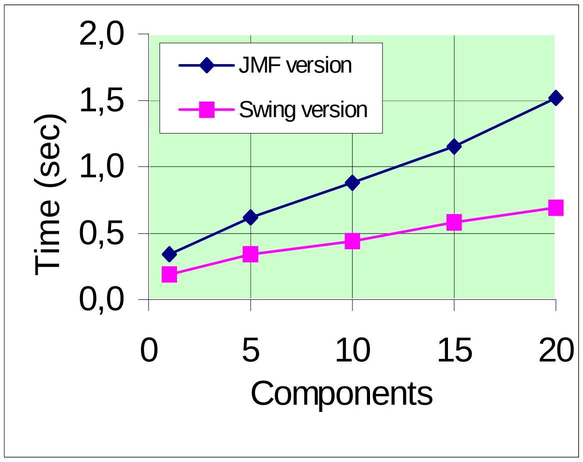


Figure 5. The time usage for prefetching one to 20 text elements.

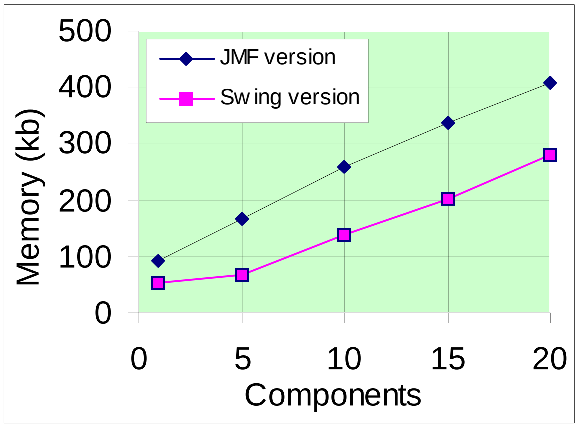


Figure 6. The memory usage for prefetching one to 20 text elements.

It can clearly be seen that the JMF implementation is slower than the Swing implementation. The JMF version is about two times slower. However, memory usage increases at the same pace in both cases. Only the initial memory usage is slightly higher in the JMF implementation.

4.2 Image component performance

Similar test were made for image elements. Again, presentations with 1, 5, 10, 15, and 20 image elements were used as a test material. Each image element referred to a 20 kb file, containing a 320x240 size image. Every image had a different URL to prevent the proxy from working. Otherwise, there would not have been any remarkable time or memory usage increase after loading the first image.

The results can be seen in Fig. 7. and Fig. 8. Once again, JMF was about twice slower than Swing. The memory usage increases at the same pace. Images allocate lots of memory, and thus the small initial difference cannot be noticed here. Each image is first decompressed in memory, which will take approximately 300 kb using total 24 bit planes for red, green, blue and alpha. The images are also double-buffered, which will result in 600 kb memory usage per image.

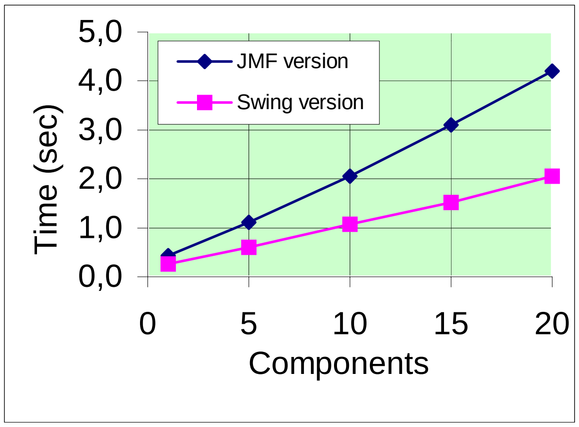


Figure 7. The time usage for prefetching one to 20 image elements.

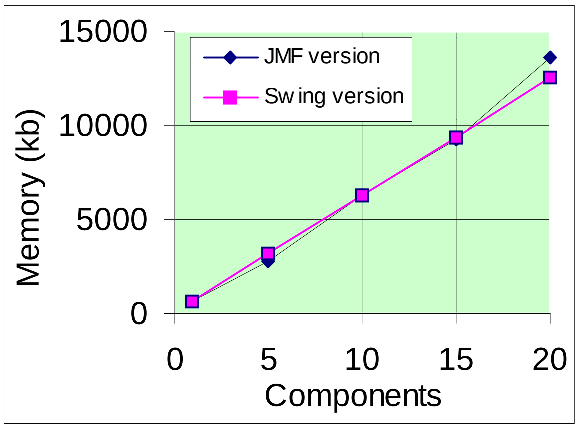


Figure 8. The memory usage for prefetching one to 20 image elements.

5 CONCLUSIONS

It is possible to implement a reasonable well performing SMIL player in Java. Processing times for simple presentations are not overwhelming. The JIT technology runs the code fast enough. However, memory handling of the media is not very efficient. The current media playback implementation allocates a lot of memory. Eventually, it will slow down the system, as the Java garbage collector will have to clean up a lot of memory. Better ways to play media could be achieved using efficient streaming. For the future, it will be a good idea to try to find solutions to play media using native methods, outside Java run-time engine. This will allow a better control over the memory and prevent long garbage collection times.

In our browser, it is easy to mix different XML languages. Also, the way the SMIL specification has been written allows displaying any imaginable media. This enables us to show richer presentations. In our case, we were able to display complete SVG files in our presentation. It is also possible to include XForms components in the presentation.

Major parts of the SMIL 1.0 specification have been implemented in X-Smiles. We will continue to support SMIL 1.0 and try to finalize the player to implement the specification as well as possible. However, the main goal in the future will be implementing SMIL 2.0 specification. This requires writing a new MLFC to the X-Smiles browser. The lessons learned in developing the SMIL 1.0 will be useful in this process.

6 REFERENCES

[1] J. van Ossenbruggen, et al., “Towards seconds and third generation web-based multimedia,” *The Tenth International World Wide Web Conference*, May 1-5, 2001, Hong Kong.

[2] P. Hoschka, “An introduction to the synchronized multimedia integration language,” *IEEE Multimedia*, vol. 5, no. 4, Oct./Dec. 1998, pp. 84-88.

[3] T. Bray et al., “Extensible markup language (XML) 1.0 (second edition),” *W3C Recommendation*, Oct. 6, 2000.

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

[5] J. Ayars et al., “Synchronized multimedia integration language (SMIL 2.0) specification,” *W3C Working Draft*, March 1, 2001.

[6] J. Ferraiolo et al., “Scalable vector graphics (SVG) 1.0 specification,” *W3C Candidate Recommendation*, Aug. 2, 2000.

[7] M. Dubinko et al., “XForms 1.0 specification,” *W3C Working Draft*, Feb. 16 2001.

[8] International Organization for Standardization/International Electrotechnical Commission. Information technology, “Coding of moving pictures and audio,” *International Standard ISO/IEC 14496:1999 (MPEG-4)*, 1999.

[9] International Organization for Standardization/ International Electrotechnical Commission. “MPEG-7: Context and objectives,” *work in progress*, 1998.

[10] D. Bulterman, et al., “GRiNS: an authoring environment for web multimedia,” *World Conference on Educational Multimedia, Hypermedia & Educational Telecommunications,* ED-MEDIA 99, Seattle, Washington USA, June 19-24, 1999.

[11] P. Vuorimaa, T. Ropponen, and N. von Knorring, “X-Smiles XML browser,” *the 2nd International Workshop on Networked Appliances*, IWNA’2000, New Brunswick, NJ, USA, Nov. 30 – Dec. 1, 2000.

[12] S. Adler et al., “Extensible stylesheet language (XSL) version 1.0,” *W3C Working Draft*, Oct. 18, 2000.

[13] Gamma et al., ”Design Patterns: Elements of Reusable Object-oriented Software.” *Addison-Wesley*, 1994.

[14] R. Gordon and S. Talley, “Essential JMF: Java Media Framework,” *Prentice Hall*, 1998.