PERFORMANCE ANALYSIS OF THE MEDIA PROCESSING COMPONENT USING XINE-LIB TO H.264 HIGH PROFILE VIDEO REPRODUCTION FOR THE GINGA MIDDLEWARE

Tiago H. Trojahn, Julio C. B. Mattos, Luciano V. Agostini,
Juliano L. Gonçalves, Leomar S. da Rosa Jr.

Federal University of Pelotas - UFPEL

ABSTRACT

Nowadays, the GingaNCL and GingaJ environments of the Ginga Middleware for the Brazilian Digital TV System don't have a way to communicate directly with each other. To solve that, a common core, named Ginga Common Core (GingaCC) is being developed in the Ginga Code Development Network project. The GingaCC will provide basic methods to both GingaNCL and GingaJ environments using different components, where each one is responsible for a specific functionality. One of the main components of the GingaCC is the one responsible to decode the video stream, named Media Processing. This work presents an implementation of the Media Processing component using the open-source Xine library and, also, presents a performance analysis in terms of processor usage and memory cost of the component implemented using various videos with different resolutions in two computer architectures.

1. INTRODUCTION

The middleware for the Brazilian Digital Television System (SBTVD), named Ginga, is an effort to create a middleware using both a declarative, known as Ginga Nested Context Language (GingaNCL) [1], and a procedural environment, known as Ginga-J [2]. The GingaNCL is based in the Nested Context Language (NCL), a declarative language developed by PUC-Rio and was recommended by International Telecommunication Union (ITU) for use in Internet Protocol Television (IPTV) systems [3]. The Ginga-J is a procedural environment built to support the Sun Java language, especially the JavaDTV package, for possible royalty's problems with the JavaTV, the standard Sun's package for iterative TV.

Actually, there are two available Ginga environments to a developer build his applications: the GingaNCL, an upgrade from a low-cost declarative middleware named Maestro built in 2001 by Moreno [4]; and the OpenGinga, an upgraded version of FlexTV [5], used nowadays as reference for the procedural middleware. Unfortunately, applications developed in GingaNCL cannot run in OpenGinga, and vice-versa.

A common core, named Ginga Common Core (GingaCC) is being developed to provide compatibility between GingaNCL and Ginga-J, forming a unique middleware for SBTVD. For this task, it was created the Ginga Code Development Network (GingaCDN) project, a network of 13 Brazilian universities, coordinated by Federal University of Paraiba (UFPB), where each university is responsible for the development of a pre-determined number of components. The Media Processing is one of these components and has a main role in GingaCC: the video decoding, part of the essential function for the video exhibition, the very fundamental requisite expected for a television system.

The Media Processing, and all others components which forms the GingaCC, will not be invoked directly by an application, but only for the Ginga,NCL and GingaJ, providing a high-level abstraction to a NCL or Java programmer.

This work was divided as follows: The section 2 presents our implementation of the Media Processing component implemented using the xine-lib, described in section 3. Section 4 describes the FlexCM, the component model used by all universities of the GingaCDN project, section 5 presents the performance tests of the component implemented and section 6 presents our conclusions and possible future works.

2. THE MEDIA PROCESSING COMPONENT

The Media Processing is the penultimate component involved in the video exhibition, working directly with the Demux and Graphics components. The components involved in video exhibition are listed as follow:

- Tuner: Component responsible for the channel tuning and the capture of the Transport Stream that is transmitted in the channel. The tuner output is redirected to the Information Service;
- Information Service: Component responsible to analyze the Transport Stream, to obtain the stream information, and to add some relevant information to reproduction;
- Demux: Component responsible to demux the streams, which compose the Transport Stream, using the information retrieved from Information Service component. The Demux output is sent to Media Processing;
- Media Processing: Component responsible to decode the stream received from Demux component. The output is sent to Graphics component;
- Graphics: The Graphics component is responsible to control and to show the decoded video in the display.

The connections among the Demux, Media Processing and Graphics components are presented in Fig. 1. The arrows represent the data flux, beginning in the Tuner and

reaching the Graphics component, passing through the Demux and Media Processing.

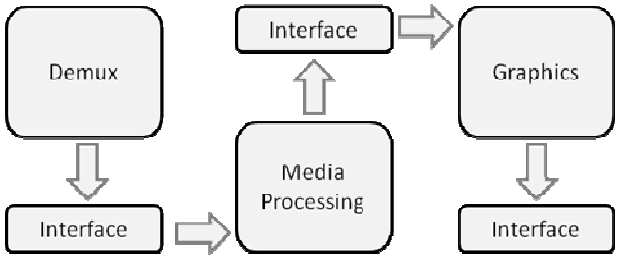

Fig. 1 – The Demux, Media Processing and Graphics connections with the correspondent interfaces and the data flux.

The Media Processing component was implemented using the C++ language with the open-source xine-lib library. To provide a standardized development and a fast integration process with others components, the FlexCM [6] component model was used. The Media Processing implementation follows the Java Media Framework (JMF) version 1.0. The JMF is a framework which provides architecture to synchronize and control video, audio and other time-based data structures, like subtitles.

The current version of the Media Processing has the following functionalities:

- Video stream decoding, including, but not limited to, the H.264/ Advanced Video Coding (AVC) formats used by SBTVD standard.
- Basic video stream flux control (play, pause and stop) methods. A seek method is not supported, because it is not foreseen in the SBTVD standard.
- Load, select and show multiple subtitle formats, like SubRip (SRT), and Advanced Substation Alpha (ASS).
- Provide a set of video stream information, like total duration, actual reproduction time, resolution, aspect ratio and Frame Rate per Second (FPS).
- Support to get screenshots and save it in a pre-determined path, with a small preview being shown in the screen.
- Support for streaming videos using Hypertext Transfer Protocol (HTTP), File Transfer Protocol (FTP), User Datagram Protocol (UDP) and Real-Time Transfer Protocol (RTP).

Only video and subtitle streams are supported by the Media Processing implementation at this time.

3. THE XINE LIBRARY

The Xine library, also known as xine-lib, is a backend library which provides audio/video demux and decoding. It was developed by Xine project under GNU General Public License (GPL) version 2. Xine is a powerful library designed to be simple and architecture independent, leaving the front-end details to other modules. The main features of the xine-lib are listed below:

- Native support for a large set of video and audio formats, like the H.264/AVC video codec and AAC audio codec, standards of SBTVD.
- Portability to all Unix-like operating systems and also to Microsoft Windows using a wrapper named w32codecs.
- Support several video drivers, like XVideo, XShm, OpenGL, X11, Xvideo, SDL, Frame Buffer and pgx64;
- The library core has developed in C language, applications in other languages can use xine-lib through dynamic library.
- Large number of possible optimizations using features like the SSE and SSE2 instruction set.

The Media Processing described in this work uses a basic and optimized X11 module created for testing purpose. The Fig. 2 illustrates some of its methods, the input stream and the X11 module.

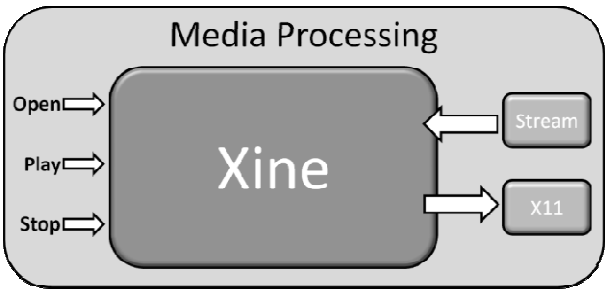

Fig. 2 – High-level description of the Media Processing implemented using the xine-lib.

The version of xine-lib library used in the implementation of the Media Processing was the 1.1.16.3 and the X11 version used was the 1.2.2.

4. THE FLEXCM COMPONENT MODEL

The GingaCDN components were developed using the FlexCM component model. Each FlexCM component must specify the required interfaces and the interfaces provided to other components. The responsibility for connecting the components is done by FlexCM during the execution time.

Each component implementation has to specify two archives:

- Architecture: This file describes the essential data for execution, like the path to the dynamic library of each component and a unique identification for the component;
- Registry: Specifies which connections are used by the component, using the unique identification numbers defined in the component implementation.

This methodology helps the distributed development needed by GingaCDN project and also guarantees an easy integration process. The version used in Media Processing implementation was the v0.2.

5. PERFORMANCE ANALYSIS

5.1. Methodology

The current version of the Media Processing component was evaluated in terms of processor percentage usage and memory cost in two personal computer architectures using data collected while playing a set of four videos in three different resolutions. The first computer, named "Computer A", is equipped with an Intel Core 2 Quad Q6600 (2.4GHz) processor and 2GB of RAM memory, the second, named "Computer B", is equipped with an Intel Core 2 Solo ULV SU3500 (1.4GHz) with 3GB of RAM memory. Both computers were running the Ubuntu 10.04 operating system.

The video set consist of four progressive videos collected across the web and encoded with the x264 revision 1376 encoder using the default settings of the High Profile and AVC at level 5.1. Are used three different videos resolutions, the 320x480 (480p), used for a large amount of mobile devices, like the Apple's IPhone and IPod, and two High Definition (HD) resolutions, the 1280x720 (720p) and 1920x1080 (1080p). The STS116 video was obtained in [7], Taxi3 French, named Park, was obtained in [8] and the Saguaro National Park, named Park, and Space Alone, named Space, was obtained in [9]. The detailed information for each resolution set is presented in Tab. 1, 2 and 3.

Tab. 1 – Detailed information for the 480p video set.

Video Name	Size (MB)	Duration (m:ss)	Average Video Bitrate (Kbps)
Park	47.8	5:20	1250
Space	27.9	3:06	1250
STS116	29.9	3:31	1250
Taxi	24.2	2:42	1250

Tab. 2 – Detailed information for the 720p video set.

Video Name	Size (MB)	Duration (m:ss)	Average Video Bitrate (Kbps)
Park	191	5:20	5000
Space	111	3:06	5000
STS116	124	3:31	5000
Taxi	96.7	2:42	5000

Tab. 3 – Detailed information for the 1080p video set.

Video Name	Size (MB)	Duration (m:ss)	Average Video Bitrate (Kbps)
Park	382	5:20	10000
Space	222	3:06	10000
STS116	248	3:31	10000
Taxi	193	2:42	10000

All videos were running rate of 30 Frame per Second (FPS), MP4 file container and doesn't have an audio track.

Three tests were performed for each video, and the samples were captured every second using the Procps application, for a time of three minutes. The component evaluated was compiled using the version 4.4.1 of GNU GCC compiler without any optimizations available in the compiler. The results for Computer A and B wasn't directly compared because of hardware differences, like available memory, hard-disk access-rate and mainboard capabilities.

The performance test results for Computer A are presented in subsection 5.2 and for Computer B in subsection 5.3.

5.2. Computer A performance evaluation

These experiments show the efficiency in terms of processor usage and memory costs of the Media Processing in a desktop. The processor, an Intel Core 2 Quad Q6600, consumes around 65.5W. It can be a reasonable consume for a set-top box since it is not a portable or battery-based device. However, this consumption can be unacceptable for most of the embedded systems, like cell phones and other mobile systems.

The processor usage, in percentage, of the component Media Processing implemented using xine-lib in Computer A is presented in Fig. 3 and the memory cost, in Megabytes (MB) is presented in Fig. 4.

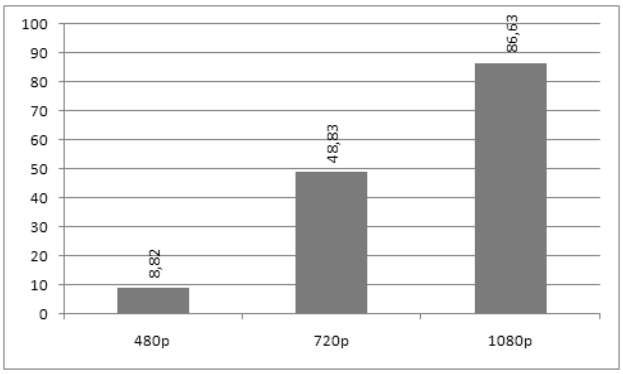

Fig. 3 – Average processor usage, in percentage, when using Computer A to execute the video set.

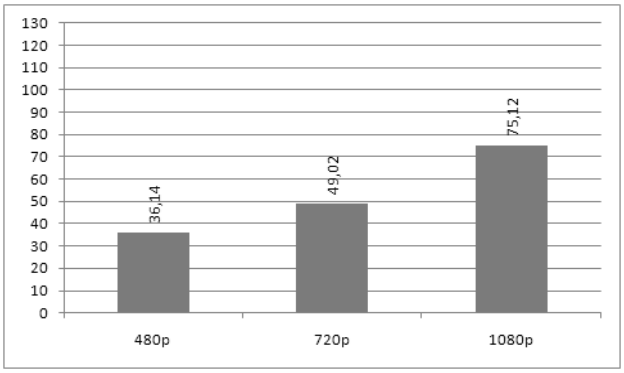

Fig. 4 – Average memory cost, in megabytes, when using Computer A to execute the video set.

In average, the processor usage of the Media Processing was 48.09%. In terms of memory, the implementation used, in average values, 53.42MB.

5.3. Computer B performance evaluation

The main goal to perform experiments in Computer B was to evaluate the efficiency of the implemented component in a personal computer with a low power consumption processor. The Intel Core 2 Solo ULV SU3500 is based on a single core architecture running at 1.4GHz, with nominal power consumption around 5.5W. This is a fundamental characteristic to embedded system and mobile devices.

The processor usage, in percentage, of the component Media Processing implemented using xine-lib in Computer A is presented in Fig. 5 and the memory cost, in Megabytes (MB) is presented in Fig. 6.

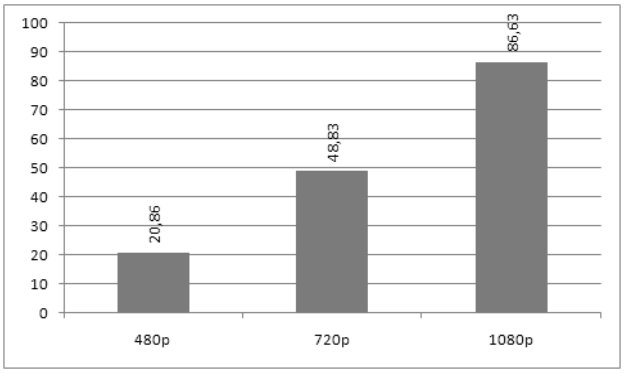

Fig. 5 – Average processor usage, in percentage, when using Computer B to execute the video set.

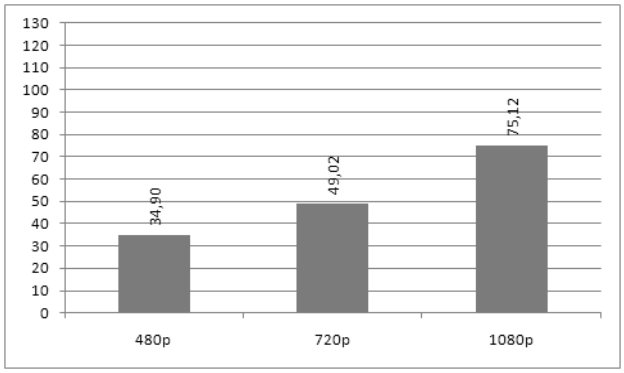

Fig. 6 – Average memory cost, in megabytes, when using Computer B to execute the video set.

In average, the processor usage of the Media Processing was 63.51%. In terms of memory, the implementation used, in average values, 53.01MB.

6. CONCLUSIONS AND FUTURE WORKS

This work presented an implementation of the Media Processing for GingaCC of the GingaCDN, using the Xine library. Experiments were performed to evaluate the behavior of the component in terms of processor usage and memory cost in two different desktop architectures.

The tests demonstrated a Media Processing component with low memory cost and processor usage. In average, the memory cost was below 55MB in average and the processor usage was below 64% in both computer architectures.

The Ginga middleware, being developed in a componentized form, will play a key role in the establishment of the Brazilian Digital TV. Its support for the environments GingaNCL and GingaJ may result in an emerging community specialized in the development of applications for both Ginga procedural and declarative sub-systems.

Regarding future works, we intend to add the support for audio streams in both Media Processing implementations and some other related features like volume control, audio track selector and so on. We will perform tests with a wider number of videos in others resolution using some optimizations in the Xine library and in the Media Processing. After all tests are completed and the component was fully tested, the integration of the Media Processing with other components of the GingaCDN project will be performed, resulting in the integrated and modular reference implementation of the Ginga middleware.

[1] Soares, L.F.G., R. F. Rodrigues, and M. F. Moreno, “Ginga-NCL: the declarative environment of the Brazilian digital TV System”. *Journal of the Brazilian Computer Society*, Porto Alegre, pp. 37-46, 2007.

[2] Filho, G.L.S., L. E. C. Leite, and C. E. C. F. Batista. “Ginga-J: The Procedural Middleware for the Brazilian Digital TV System”. *Journal of the Brazilian Computer Society*, pp.47-56, 2007.

[3] ITU, “ITU-T AAP Recommendation H.761: Nested Context Language (NCL) and Ginga-NCL for IPTV”, Switzerland, 2009.

[4] Moreno, M.F. *A declarative middleware for Interactive Digital TV Systems*, M.S Thesis, PUC-Rio, Rio de Janeiro, 2006.

[5] Ferreira, C.L.P., *Maestro: A middleware to supports distributed applications based in software components*. M.S Thesis, USP, São Paulo, 2001.

[6] Filho, S. M.S., L.E.C. Leite, G. Lemos, and S. Meira, "FLEXCM - A Component Model for Adaptive Embedded Systems", *31st Annual International Computer Software and Applications Conference*, Beijing, pp. 1-10, 2007.

[7] Nasa, “NASA High Definition Video”, nasa.gov, Dec. 16, 2009 [Online]. Available: http://www.nasa.gov/ multimedia/ hd/index.html, [Accessed: June 11, 2010].

[8] Microsoft, “WMV HD Content Showcase”, microsoft.com, 2004 [Online]. Available: http://www. microsoft.com/windows/ windowsmedia/musicandvideo/hdvideo/contentshowcase.aspx, [Accessed: June 11, 2010].

[9] Adobe, “Adobe Flash HD Gallery”, adobe.com, July 14, 2009 [Online]. Available: http://www.adobe.com/mena/ products/hdvideo/hdgallery/, [Accessed: June 11, 2010]