A Comparative Analysis of Media Processing Component Implementations for the Brazilian Digital TV Middleware

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

Federal University of Pelotas, Informatics Department
Group of Architectures and Integrated Circuits
{tiagot.ifm, julius, agostini, juliano.lucas, leomarjr}@ufpel.edu.br

Abstract—In the Brazilian Digital Television System (SBTVD) two environments are defined, the procedural and the declarative one, which run simultaneously in the same middleware. The procedural environment is called GingaJ and uses the Sun Java language, while the declarative one is called GingaNCL and it is based on the Nested Context Language (NCL). These two environments compose the Ginga middleware. Nowadays, there is not a single middleware which provides support for both GingaJ and GingaNCL environments. For this reason, the Ginga Code Development Network (GingaCDN) was created to design and provide a complete, tested, integrated and componentized reference version of the middleware. Thus, a common core, named Ginga Common Core (GingaCC) is being developed to provide support for both environments. Solutions are being investigated for the media processing, one of the main GingaCC components, responsible to decode video, audio and subtitles streams. In this work, two media processing implementations are presented, one based on libVLC library and other based on Xine, in order to explore different solutions. This paper discusses both implementations and compares its efficiency through experiments using different computer architectures.

Keywords—Middleware; digital TV; Ginga; Media processing; libvlc.

I. INTRODUCTION

In the middle of 2003, the Brazilian Digital Television System (SBTVD, short for “Sistema Brasileiro de TV Digital”) was created by the government decree 4901 [1] to encourage the creation of a national Digital TV standard. In 2006, the Japanese standard, the Integrated Services Digital Broadcasting-Terrestrial (ISDB-T), was considered that most closely met national needs in a study [2] published by Brazilian Society for Television Engineering (SET, short for “Sociedade Brasileira de Engenharia de Televisão”) and the Brazilian Association of Radio and Television Broadcasters (ABERT, short for “Associação Brasileira de Emissoras de Rádio e Televisão”). Thus, the ISDB-T was officially adopted by SBTVD in the government decree 5820 [3].

Also, in 2006, Federal University of Paraiba (UFPB, short for “Universidade Federal da Paraíba”) developed a middleware, named FlexTV [4], to follows the specifications of the European Globally Executable MHP (GEM) [5]. From the FlexCM, the OpenGinga middleware was conceived. This middleware has the capabilities to run some programs based in JavaDTV [6], a Java package especially developed by Sun Microsystems based in the JavaTV but without royalties’ problems. That effort was recently named GingaJ [7].

In 2001, Ferreira [8] proposed the creation of a low-cost declarative middleware. Moreno [9] implemented the Maestro middleware using the Nested Context Language (NCL). The NCL is a declarative language developed by PUC-Rio focused in the media synchronism and, highlighted by Ferreira, known by its low computer cost. In 2007, the name Maestro was changed to GingaNCL and was officially adopted as the declarative environment of SBTVD [10]. The GingaNCL was recommended, in late 2009, by International Telecommunication Union (ITU) for use in Internet Protocol Television (IPTV) systems [11].

This separated development reflected in a disjoint between the declarative and the procedural environment. Applications developers have to choose one of the environments and cannot use the features of the other paradigm, making the software restrict to only one middleware implementation.

Finally, in February 2009, it was created the GingaCDN Project, financed by the RNP Brazilian Funding Agency. The aim of GingaCDN [12] is create a distributed and collaborative network of software development for Digital TV conotributing to the creation of a single, modular and integrated reference implementation of Ginga, unifying the GingaNCL and the GingaJ environment.

The middleware has been developed using a component-based approach, introducing a common core, named Ginga Common Core (GingaCC), to provide support for the declarative GingaNCL and the procedural GingaJ sub-systems. The software development process is being done by 13 Brazilian universities, coordinated by UFPB, where each university is responsible for the design of a pre-determined number of components. This effort includes the creation of a tool to make custom versions of Ginga, where a developer can combine different components, with different features, to build a custom Ginga middleware. Every GingaCC components are being developed using the C++ language and using a component model named FlexCM [13] to facilitate the integration of all components.
The aim of this paper is the media processing component implementation, which is one of the main components of the GingaCC. To explore different solutions, two components with the same behavior were developed. The first solution is based on the libVLC library [14], while the second one is based on the Xine library [15]. This work presents a comparison between both solutions.

The media processing is described in the section II. The libraries used in the implementation, the libVLC, and Xine, as well as the FlexCM component model are described in the subsections A, B and C, respectively. The section III presents the Experimental Results and section IV presents the conclusions and future works.

II. MEDIA PROCESSING

GingaCC is responsible to provide support to GingaNCL and GingaJ applications, through a set of fundamental functionalities. One of these basic functionalities is the video stream decoding, which ultimately leads to the video display. In the GingaCC, the component responsible by this function, the video decoding, is the media processing.

The current version of the media processing have a set of method which perform the basic functionalities, like video decoding, and some others directly related. A high-level description of these functionalities is described as follow:

- 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).

All media processing functions are based in the Java Media Framework (JMF) version 1.0 [16]. The JMF is an Application Programming Interface (API) which specifies a set of methods and use cases to synchronize, control, process and presents audio, video, subtitles and other time-based data structures.

In the GingaCC, the media processing interacts directly to other two components, the Demux and the Graphics, described as follow:

- Demux: responsible of separating the various streams present in the Transport Stream and to provide it to the correct component to be processed.
- Graphics: Component responsible to show the video and subtitles in the display. Receives the decoded stream from media processing and processes it with a built-in video output driver.

The interfaces and the connections among the Demux, media processing, and Graphics components of the GingaCC are depicted in Fig. 1.

In order to explore the media processing component efficiency, we have developed two implementations of this component. The first one is implemented in libVLC and offers all functionalities described above. For testing purposes, the second implementation uses the Xine backend library and an optimized X.Org module to show the resulting decoded video and only provides the methods to open, decode and play an input video stream.

A. LibVLC

The libVLC library [14] 1.0.6 version was used in the implementation of the media processing. LibVLC is a graphic library developed by VideoLAN under GNU General Public License (GPL) version 2. The choice for this library to implement the media processing component is due to its large list of features, like:

- Compatibility with various media formats, including the standard H.264/AVC of the SBTVD, the audio standard MPEG Layer 2, MPEG Layer 3 (MP3) and MPEG-4 part 3 AAC (Advanced Audio Coded);
- Portability to a wide range of operating systems, like Microsoft Windows, GNU Linux, Mac OS, BeOS and FreeBSD;
- Support for several video outputs, like DirectX, OpenGL, X11, Xvideo, SDL and Frame Buffer;
- The library itself is written in C language, offering a high performance to process a media. The library was, also, ported to other languages, like Java.
- Have an extensive API, providing demux, video display and exception control.

In the highest abstraction level, the libVLC library presents two complementary classes, libVLC_media_player and media processing.
and *libVLC_media*. The main functions of those classes are explained as follow:

- **libVLC_media_player** – provides methods to control the reproduction and to return information from the stream flow. There are also some other functions, like append subtitles to stream;
- **libVLC_media** – provides low-level methods that allow control the media, like the descriptor duplication, calculus of duration for the media being used and various meta information. It can be separated in *libVLC_video* and *libVLC_audio*, the lowest level class which controls video and audio, respectively.

A high-level description of some methods of *libVLC_Media_Player* and *libVLC_Media* implemented in the media processing are shown in Fig. 2.

**B. Xine**

Xine library [15], 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 as a multi-thread media player, originally designed to provide an easy graphical DVD reproduction. 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.
- Support several video drivers, like XVideo, XShm, OpenGL, X11, Xvideo, SDL, Frame Buffer and pgx64;
- The library core was developed in C language, applications in other languages can use Xine-lib through dynamic library.

The media processing described in this work uses a basic and optimized X.Org module created for testing purpose. Fig. 3 illustrates some of its methods, the input stream and the X.Org module.

**C. FlexCM**

GingaCDN components have been developed using the FlexCM component model [13]. Each FlexCM component must specify the required interfaces and, also, the interfaces provided to other components. The responsibility for connecting the components is done by FlexCM during the execution time.

For each component implementation one should also specify two archives, Architecture and Registry. The first one describes the data for execution, like the path to the dynamic library of each component and a unique identification for the component. The Registry file specifies which connections are used by the component, using the unique identification numbers defined in the component implementation.

This methodology aids the distributed development and also guarantees an easy integration process. The FlexCM version used in media processing implementations was the 0.2.

**III. EXPERIMENTAL RESULTS**

**A. Methodology**

The experiments were performed in three different computer architectures, a desktop named “Computer A”, a notebook named “Computer B” and other desktop, named “Computer C”. The Computer A is an Intel Core 2 Duo E6320 1.86 GHz processor, with 2 Gigabytes (GB) of RAM, running the Ubuntu 10.04 operating system. Computer B is an Intel Core 2 Solo ULV SU3500 1.40 GHz processor, with 3GB of RAM and running the Ubuntu 10.04 operating system and the Computer C is an Intel Core 2 Quad Q6600 2.4 GHz processor with 2GB of RAM and running the Ubuntu 10.04 operating system.

The experimental data was obtained through analysis of the processes, using the Procps application to capture these data. Each benchmark video was executed during three minutes, being repeated three times. The memory cost and the processor usage of the media processing were captured each second while running the video stream, resulting in 540 samples for each video. A total of 6480 samples were captured.
analyzed for each library, resulting in 19440 samples, considering the three computer architectures. Five percent of the greatest values and five percent of the lowest values registered in processor usage and memory cost were removed to prevent outliers which would influence the final results. This methodology provides a consistent performance analysis of the media processing in real use. However, only video stream reproduction was analyzed, because the lack of other features, like screenshots and video resizing in the media processing implemented with the Xine library.

The benchmark set consists of four progressive videos in three different resolutions, the 848x480 (480p) known as Standard Definition (SD), the 1280x720 (720p) and the 1920x1080 (1080p) known as High Definition (HD). The STS116 video was obtained from [17], the Taxi3 French, named Taxi, was obtained from [18] and the Saguaro National Park, named Park, as well as the Space Alone, named Space, were available in [19]. The video details are presented in Table 1 (480p videos), Table 2 (720p videos), and Table 3 (1080p video). All videos have 16:9 as aspect ratio, MP4 container, 30 FPS and do not have audio track.

<table>
<thead>
<tr>
<th>Video Name</th>
<th>Size (MB)</th>
<th>Duration (m:ss)</th>
<th>Video Bitrate (Kbps)</th>
<th>Resolution (pixel X pixel)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Park</td>
<td>95.7</td>
<td>5:20</td>
<td>2500</td>
<td>848x480</td>
</tr>
<tr>
<td>Space</td>
<td>55.7</td>
<td>3:06</td>
<td>2500</td>
<td>848x480</td>
</tr>
<tr>
<td>STS116</td>
<td>62.1</td>
<td>3:31</td>
<td>2500</td>
<td>848x480</td>
</tr>
<tr>
<td>Taxi</td>
<td>48.5</td>
<td>2:42</td>
<td>2500</td>
<td>848x480</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Video Name</th>
<th>Size (MB)</th>
<th>Duration (m:ss)</th>
<th>Video Bitrate (Kbps)</th>
<th>Resolution (pixel X pixel)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Park</td>
<td>191</td>
<td>5:20</td>
<td>5000</td>
<td>1280x720</td>
</tr>
<tr>
<td>Space</td>
<td>111</td>
<td>3:06</td>
<td>5000</td>
<td>1280x720</td>
</tr>
<tr>
<td>STS116</td>
<td>124</td>
<td>3:31</td>
<td>5000</td>
<td>1280x720</td>
</tr>
<tr>
<td>Taxi</td>
<td>96.7</td>
<td>2:42</td>
<td>5000</td>
<td>1280x720</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Video Name</th>
<th>Size (MB)</th>
<th>Duration (m:ss)</th>
<th>Video Bitrate (Kbps)</th>
<th>Resolution (pixel X pixel)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Park</td>
<td>382</td>
<td>5:20</td>
<td>10000</td>
<td>1920x1080</td>
</tr>
<tr>
<td>Space</td>
<td>222</td>
<td>3:06</td>
<td>10000</td>
<td>1920x1080</td>
</tr>
<tr>
<td>STS116</td>
<td>248</td>
<td>3:31</td>
<td>10000</td>
<td>1920x1080</td>
</tr>
<tr>
<td>Taxi</td>
<td>193</td>
<td>2:42</td>
<td>10000</td>
<td>1920x1080</td>
</tr>
</tbody>
</table>

The videos were codified from the 1080p video sources, using the x264 version 1376 library [20]. The H.264 High profile and AVC at level 5.1 were used with a constant bit rate for each resolution. The frame rate of all videos was converted to 30 FPS. Some of the important coding details are listed below. They are all default to High Profile of the x264 encoder at AVC 5.1 level:

- Context-adaptive binary arithmetic coding (CABAC) enabled;
- Deblocking enabled, strength and threshold are 0;
- Motion Estimation (ME) hexagonal algorithm with range 16, using Chroma M.E and RDO on all I/P frames for Subpixel Refinement;
- Macroblocks with adaptive Discrete Cosine Transform (DCT) using I4x4, I8x8, P8x8 and B8x8;
- Trellis disabled;
- Three B-Frames with bias equals to 0. Fast search for adaptive B-Frames and B-Pyramid disabled;
- Three reference frames with Adaptive I-Frame Decision enabled were used;
- Colorimetry at 4:2:0.

B. Analysis of Results for Computer A

These experiments show the efficiency in terms of processor usage and memory costs of the media processing in a mid-end desktop. The processor, an Intel Core 2 Duo E6320, consumes around 65.5W [21] just for the processor. 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, achieved for both Xine and libVLC implementations of media processing using benchmark with different resolutions and running in Computer A is shown in Fig. 4.

![Figure 4. Average processor usage for Xine and libVLC implementations of the media processing, in percentage, when using Computer A.](image-url)
Figure 5. Average memory cost for Xine and libVLC implementations of media processing, in Megabytes, when using Computer A.

The memory cost variation, in percentage, between libVLC and Xine implementations for 480p videos was 67.17%, for 720p videos was 48.45%, and for 1080p videos was 34.26%. The memory cost of the libVLC implementation presented a greater memory cost in all video resolutions in comparison with the Xine implementation of the media processing.

C. Analysis of Results for Computer B

The main goal to perform experiments in Computer B was to evaluate the efficiency of the two implemented components 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 [22] just for the processor. This is a fundamental characteristic to embedded system and mobile devices.

The processor usage, in percentage, for both Xine and libVLC implementations of media processing in Computer B are illustrated in Fig 6.

Figure 6. Average processor usage for Xine and libVLC implementations of the media processing, in percentage, when using Computer B.

The processor usage variation, in percentage, between libVLC and Xine implementations for 480p videos was 1.21%, for 720p videos was 1.55%, and for 1080p videos was 2.5%. The Xine implementation presented a greater processor usage in 720p and 1080p videos.

The memory costs, in Megabytes, of the libVLC and Xine implementations of the media processing are presented in Fig. 7.

Figure 7. Average memory cost for Xine and libVLC implementations of media processing, in Megabytes, when using Computer B.

The memory cost variation, in percentage, for 480p videos was 28.99%, for 720p videos was 20.74%, and for 1080p videos was 34.26%. The libVLC implementation presented a larger memory consume in all video resolutions in comparison with the Xine implementation of the media processing.

D. Analysis of Results for Computer C

These experiments show the efficiency in a quad-core environment. The processor, an Intel Core 2 Quad Q6600, consumes around 105W [23] just for the processor, unacceptable for most embedded systems, like cell phones.

The processor usage, in percentage, for both Xine and libVLC implementations of media processing in Computer C are illustrated in Fig 8.

Figure 8. Average processor usage for Xine and libVLC implementations of the media processing, in percentage, when using Computer C.

The processor usage variation, in percentage, between libVLC and Xine implementations for 480p videos was 4.20%, for 720p videos was 7.24%, and for 1080p videos was 3.86%. The Xine implementation presented a greater processor usage in all videos resolutions in comparison with the libVLC implementation of the media processing.

The memory costs, in Megabytes, of the libVLC and Xine implementations of the media processing are presented in Fig. 9.

Figure 9. Average memory cost for Xine and libVLC implementations of media processing, in Megabytes, when using Computer C.
The memory cost variation, in percentage, for 480p videos was 31.29%, for 720p videos was 23.70%, and for 1080p videos was 19.39%. Again, the memory cost of the libVLC implementation presented a greater memory cost in all video resolutions in comparison with the Xine implementation of the media processing.

E. General Results

The following results consider the average values for all video resolutions, obtained for both implementations.

Fig. 10 presents the average processor usage for the libVLC and Xine implementations of media processing, in percentage, obtained using Computer A. In this case, the average processor usage variation, in percentage, was 1.34%.

The average memory cost with libVLC and Xine implementations obtained using Computer A is shown, in Megabytes, in Fig. 11. In this case, the average memory cost variation, in Megabytes, was 45.51%.

Fig. 12 presents the average processor usage for the libVLC and Xine implementations of media processing, in percentage, obtained using Computer B. The average processor usage variation, in percentage, was 1.38%.

The average memory cost with libVLC and Xine implementations obtained using Computer B is shown, in Megabytes, in Fig. 13. In this case, the average memory cost variation, in Megabytes, was 21.39%.

Fig. 14 presents the average processor usage for the libVLC and Xine implementations, in percentage, obtained using Computer C. In this case, the average processor usage variation, in percentage, was 4.94%.
The average memory cost with libVLC and Xine implementations obtained using Computer C is shown, in Megabytes, in Fig. 15. In this case, the average memory cost variation, in Megabytes, was 23.27%.

The experiments among Computer A, Computer B and Computer C cannot be directly compared, because the differences of the hardware utilized, like main boards, memory available, hard-disk access rate, and also for the different processes running in each operating system. However, the results demonstrate that in terms of processor usage the libVLC implementation can achieve better results. On the other hand, in terms of memory usage the best choice is the Xine implementation, which presents smaller memory needs.

It is important to emphasize that tests presented here are preliminary. In a near future, we are going to perform another set of tests using a wider range of videos with some other resolutions to evaluate the components. Also, is being studied another metric to analyze the components, in addition to processor usage and memory cost presented in this work.

IV. CONCLUSIONS AND FUTURE WORKS

This paper presented two implementations of the media processing for GingaCC using the libVLC and the Xine libraries. Experiments were performed to evaluate the behavior in terms of processor usage and memory cost of the two components in two different architectures.

The experimental results show a better performance to the media processing implemented with the libVLC library when considering processor usage. The processor usage of the media processing implemented with the Xine library was 1.49% greater than the media processing implemented with libVLC. In terms of memory cost, the implementation using Xine library showed a better performance. It presented 45.51%, 21.39% and 23.27% memory usage reduction if compared to the libVLC implementation when running in Computer A, Computer B and Computer C, respectively.

The flexibility of Ginga, being developed in a componentized form, will play a key role in the establishment of the Brazilian Digital TV System. Its support for the environments GingaNCL and Ginga 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. After that, 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.

REFERENCES


