An Android Multimedia Framework based on Gstreamer

Hai Wang[1], Fei Hao[2], Chunsheng Zhu[3],
Joel J. P. C. Rodrigues[4], and Laurence T. Yang[3]

[1] School of Computer Science, Wuhan University, China
hkhaiwang@gmail.com
[2] Department of Computer Science, KAIST, South Korea
fhao@kaist.ac.kr
[3] Department of Computer Science, St. Francis Xavier University, Canada
{chunsheng.tom.zhu,ltyang}@gmail.com
[4] Instituto de Telecomunicações, University of Beira Interior, Portugal
joeljr@ieee.org

Abstract. Android is a widely used operating system in mobile devices, due to that it is free, open source and easy-to-use. However, the multimedia processing ability of current android is quite limited, as the original android multimedia engine OpenCore cannot deal with lots of commonly used video (audio) formats. Recently, several approaches are proposed to enhance the multimedia processing ability and Gstreamer based method is supposed to own the best performance. However, the multimedia processing ability of current extension multimedia frameworks are still not good enough, which weakens the potential application prospect. In this paper, we provide another android multimedia framework based on Gstreamer. Extensive experiments show that our Gstreamer based framework can greatly improve the multimedia processing ability in terms of efficiency, compatibility, feasibility and universality.

Key words: Android, Multimedia framework, OpenCore, Gstreamer

1 Introduction

Multimedia supported green mobile networks can offer a lot of benefits for people [1]. Released by Google and supported by OHA (Open Handset Alliance), android is a widely used open source operating system for mobile devices. Apart from its operating system character, android is also a mobile software development platform which includes operating system kernel, application framework and core applications. Because it is free, open-source and easy-to use for both application developers and users, many developers and users have converted to it and it has a very bright future in the mobile market [2] [3] [4] [5]. Moreover, many multimedia terminals such as Google TV and iPad-like terminals have been popular in recent years and android OS can been modified and ported to be applied into them. This further extends the market prospect of android.

However, as the original multimedia engine OpenCore cannot deal with lots of commonly used video (audio) formats, the multimedia processing ability of android is quite limited and it cannot satisfy the various multimedia processing demand imposed by multimedia terminal devices. Recently, several approaches are put forward to enhance the multimedia processing ability of android. Specially, [6] proposes to add some audio/video coding (encoding) libraries into the OpenCore engine as plug-ins to improve the processing ability of OpenCore. [7] and [8] try to extend the Java application framework with NDK development method to perform more functions. [9] intends to employ the Gstreamer multimedia engine to supply more multimedia services for the application client. Among these extension methods, the Gstreamer based method is supposed to be the most effective, as Gstreamer is a popular multimedia engine with rich plug-ins. Whereas, the multimedia processing ability of current android multimedia frameworks are still not good enough. For example, many frameworks can only deal with specific video (audio) formats. In this paper, another design of an android multimedia framework based on Gstreamer which greatly enhances the multimedia processing ability is presented. Extensive experiments are conducted and they show that our framework obtains high efficiency, compatibility, feasibility and universality. To the best of our knowledge, our work provides essential contribution for further research regarding Gstreamer based multimedia frameworks and their commercial applications.

For the rest of this paper, section 2 briefly introduces the system architecture of android. The original android multimedia framework is discussed in section 3. Our extended Gstreamer based multimedia framework is described in section 4. Section 5 shows the experiments and gives experimental results analysis. And section 6 concludes this paper.

2 Android System Architecture

The android system consists of five layers and each layer consists of some core components. Figure 1 shows the architecture of android. From top to down, the core components are: Applications, Application Framework, Native C libraries, Android Runtime Environment (JVM), HAL (Hardware Abstract Layer), Linux Kernel.

1) Applications. Application layer consists of many core Java-based applications, such as calendar, web browser, SMS application, E-mail, etc.

2) Application Framework. Application framework consists of many components and Java classes to allow android application developers to develop various kinds of applications. By using Java language, it hides the internal implementation of system core functions and provides the developers an easy-use API. Basically, it includes Java core class and some special components of android. Some typical components are as follows: View (List, Grids), Content Provider, Resource Manager, Activity Manager.

3) Native C Libraries. In Native C library layer, it consists of many C/C++ libraries. And the core functions of android are implemented by those libraries.

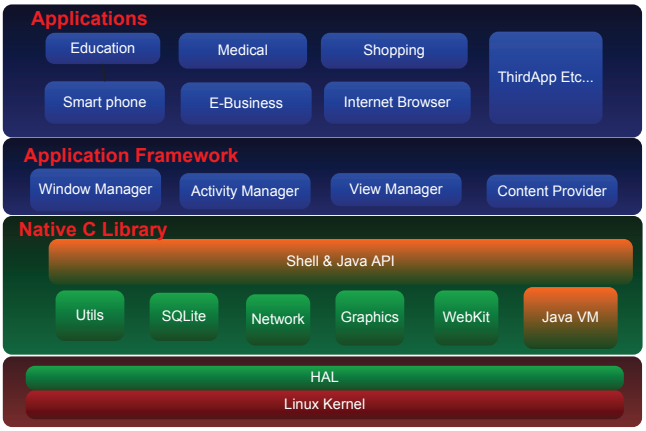

Fig. 1. Android architecture

Some typical core libraries are as follows: Bionic C lib, OpenCore, SQLite, Surface Manager, WebKit, 3D library.

4) Android Runtime Environment. Runtime environment consists of Dalvik Java virtual machine and some implementations of Java core libraries.

5) HAL. This layer abstracts different kinds of hardwares and provides an unified program interface to Native C libraries. HAL can make Android port on different platforms more easily.

6) Linux Kernel. Android's core system functions (e.g., safety management, RAM management, process management, network stack) depend on Linux kernels.

3 Android Multimedia Framework

3.1 Overall Multimedia Architecture

The android multimedia system includes multimedia applications, multimedia framework, OpenCore engine and hardware abstract for audio/video input/output devices. And the goal of the android multimedia framework is to provide a consistent interface for Java services. The multimedia framework consists of several core dynamic libraries such as libmediajni, libmedia, libmediaplayservice and so on [4].

A general multimedia framework architecture is shown in Figure 2. From Figure 2, we can see that, Java classes call the Native C library Libmedia through Java JNI (Java Native Interface). Libmedia library communicates with Media Server guard process through Android's Binder IPC (inter process communication) mechanism. Media Server process creates the corresponding multimedia service according to the Java multimedia applications. The whole communication between Libmedia and Media Server forms a Client/Server model. In Media

Server guard process, it calls OpenCore multimedia engine to realize the specific multimedia processing functions. And the OpenCore engine refers to the PVPlayer and PVAuthor.

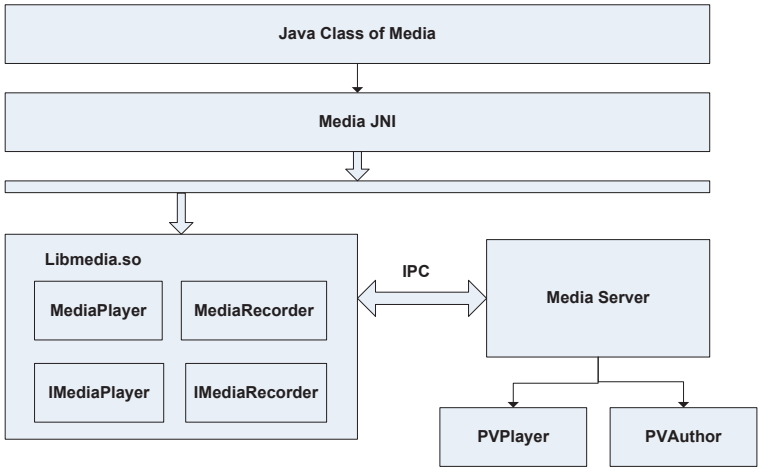

Fig. 2. Android multimedia framework architecture

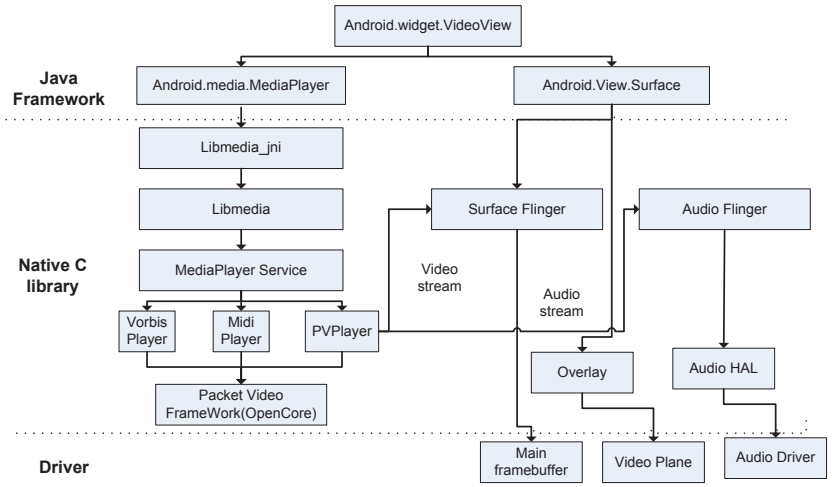

Fig. 3. Details of android multimedia framework architecture

More detailed information regarding the multimedia framework are shown in Figure 3. From Figure 3, we can see that the typical video/audio data stream works in Android as follows. Specially, Java applications first set the URI of

the media (from file or network) to PVPlayer through Java framework, JNI and Native C. In this process, there are no data stream flows. Then PVPlayer processes the media data stream with the following steps: demux the media data to sperate video/audio data stream, decode video/audio data, sync video/audio time, send the decoded data out.

3.2 OpenCore Multimedia Engine

OpenCore is the core of the android multimedia system. Generally speaking, it has the following characters. First, it should support most common audio formats and support stream media (RTSP/RTP). Second, it should be extended with the third-party Codecs. A general architecture of OpenCore is described in Figure 4.

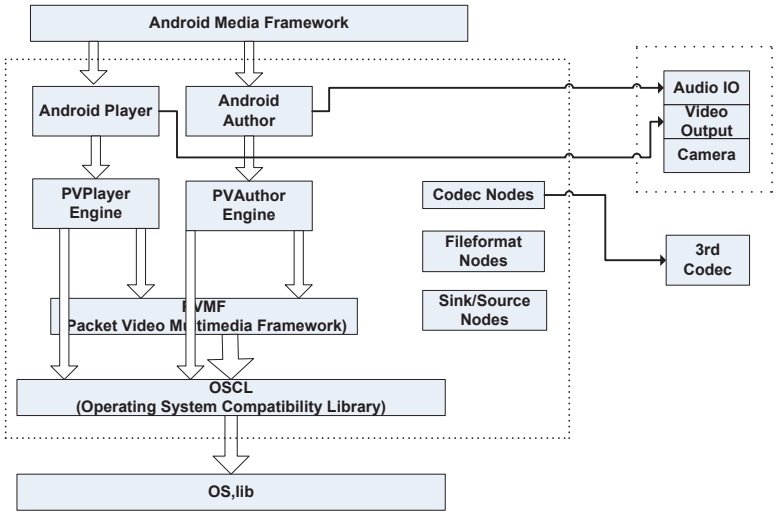

Fig. 4. OpenCore architecture

From Figure 4, we can see that OpenCore owns many functions such as media file format analysis, audio/video decoding and so on. From down to top, OpenCore includes OSCL (Operating System Compatible Layer), PVMF (Packet Video Multimedia Framework), File Formats analysis Node, Decoding Node, Encoding Node, Media I/O Node, Player engine. Furthermore, we can see that PVPlayer used in multimedia framework calls Player engine in OpenCore to realize the specific functions. At the same time, we can find that we can integrate third party Codecs into OpenCore through adding new decoding Node.

4 Gstreamer based Multimedia Framework

4.1 Design of the Multimedia Framework

Gstreamer is a popular and widely used multimedia engine in Linux and many commercial media players use it as the core kernel [10]. The basic idea of Gstreamer based approach is to port Gstreamer to android and it is first proposed in [9]. Compared with other extension methods (e.g., [6] [7] [8]), porting Gstreamer to android has the following advantages. First, it extends the Android's multimedia processing ability in the Native C without changing the Java API, so all the Java applications can benefit from this extension without any modification. This can avoid the limited Java application problem compared with the methods in [7] [8]. Second, Gstreamer is widely used, so there are lots of available popular plug-ins. If a new function is needed, we only need to add the corresponding plug-ins to Gstreamer. This can significantly reduces the work of extension compared with the approach proposed in [6]. Third, Gstreamer owns OpenMax IL Standard plug-in instead of using software to process video/audio. Thus, it can realize hardware video processing acceleration on the development board which supports OpenMax IL. With Gstreamer, Android can make use of the ability of hardware to the uttermost [11] [12] [13] [14].

As for our Gstreamer based multimedia framework, first, we add Gstreamer to the Media Server guard process, thus the Media Server guard process can provide multimedia play service to Java applications. Second, we change the compilation part of Gstreamer so that the total multimedia framework can run on different kinds of multimedia terminals with CPU architectures (e.g., X86, ARM, MIPS, SH4) [15]. Third, we employ the Prelink technology to enhance the execution efficiency. Last, we also apply the OpenMax IL Standard, thus our approach can support hardware video processing acceleration in special development boards. Figure 5 shows our extension method with Gstreamer.

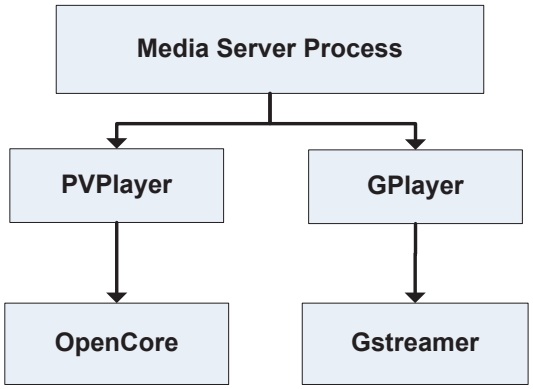

Fig. 5. Our Gstreamer-based multimedia framework architecture

To the best of our knowledge, current android multimedia frameworks exploiting Gstreamer are almost all based on the first Gstreamer based android multimedia framework in [9]. Here, different from the Gstreamer multimedia framework in [9], our framework has the following advantages. First, we still keep OpenCore instead of replacing OpenCore, as OpenCore is more sufficient than Gstreamer when dealing with H.264 encoded videos. Thus our framework can take advantage of both OpenCore and Gstreamer. Second, we modify the assembly languages in Liboil and Gstreamer engine so that our framework can run on different CPU architectures. Third, by adding more sufficient plug-ins, our framework can deal with more video (audio) formats and it has a high compatibility to different Video/Audio clips. Finally, we make some performance optimizations due to that Gstreamer has some performance weaknesses under embedded environment.

4.2 Implementation of the Multimedia Framework

To achieve the desirable functions, we need to the following work.

1) Porting some Gstreamer required open-source libraries to Android, such as Glib, Liboil, etc.

2) According to Gstreamer framework, write two plug-ins. One is used to send the decoded original video data from Gstreamer to Android display system (Surfaceflinger). And another is used to send the decoded original PCM audio data to Android audio system (Audioflinger).

3) Based on Gstreamer, we should construct a mediaplayer which can be used in Media Server guard process to supply media player service.

4) Modify assembly codes to apply Gstreamer to diversified CPU architectures, and apply some commonly used optimized technologies in embedded environment to Gstreamer.

As the porting work of Glib, Libiol and other related libraries are easy, here we introduce the rest steps.

1) The Surfaceflinger. Surfaceflinger is an important part of Android's display system. And its specific implementation mechanisms are quite complex. Moreover, it provides programming interfaces to Native C users and Java users. To send our decoded raw data to Surfaceflinger to display, in the native C user, we can do the following.

First, create a display layer through Surfaceflinger Client and get the control interface, namely, ISurface interface. Second, open the Session between Surfaceflinger and the ISurface Interface. Third, utilize ISurface to control the Layer's attributes, such as Alpha, Z-order, then close the session. Fourth, clear the previous display buffer, then post the current buffer to the bufferheap in ISurface. Last, destroy display buffer and unregister ISurface interface.

The core codes for implementation are described as follows.

2) The Audioflinger. Audioflinger is an important part of Androids audio system. Meanwhile, it provides programming interfaces to Native C users and Java users. To send our decoded raw data to Audioflinger to play, in the native C user, we need to do the following.

Algorithm 1 Core Codes for Implementation

```
1: SurfaceComposerClient client= new SurfaceComposerClient();
2: client.Register();
3: OpenSession();
4: ISurface control=client.get();
5: control.setAlpha();
6: control.setLayer();
7: CloseSession();
8: control.clear();
9: control.BufferHeap= SurfaceSink.get().BufferHeap;
10: control.Post(BufferHeap);
11: client.UnRegister();
```

First, create an AudioTrack through Audioflinger Client and obtain the control interface, namely, ISurface interface. Second, open the Session between Audioflinger and the ISurface Interface. Third, utilize ISurface to control the attributes of the Layers, such as FrameCount, Audio Format, then close the session. Fourth, clear the previous audio buffer, then post the current buffer to the bufferheap in ISurface. Last, destroy display buffer and unregister ISurface interface.

The core codes for this part are almost the same with that of Surfaceflinger in Algorithm 1.

3) The Mediaplayer. After we build all the essential Gstreamer plug-ins, we need to use Gstreamer to construct a media player inherited from Android MediaPlayer interface to provide media processing service for Java framework. In those steps, we can use some low-level elements in Gstreamer to construct mediaplayer. But this method is extremely complex. For example, regarding a specific video clip, we need to analyze which video container format it belongs to. Then according to different formats, we need to prepare different demux, decode elements, and connect all the essential elements in a pipeline to let the whole video processing steps run in it. As different video clips correspond different processing elements, for practical use and stability, we use the Gstreamer existed high-level element "Playbin2" to start the pipeline. Then we add a bus to Playbin2 to let the video data and video processing event flow in the bus. Finally, we start the pipeline and bus. During implementation, we use gst-ffmpeg and gstopenmax plug-ins to decode video/audio data. When running our framework, the Gstreamer sends the original video data (RGB 555) to Surfaceflinger, and sends original audio data to Audioflinger. As for media record service, media meta information service and play service for H.264 encoded videos, we still use OpenCore because OpenCore can provide the whole essential functions, there is no need to extend it. Figure 6 shows the whole system after our extension.

4) Optimization methods. Due to Gstreamer is mainly used in Linux for PC, for usage in diversified embedded environment, we need to do some optimizations.

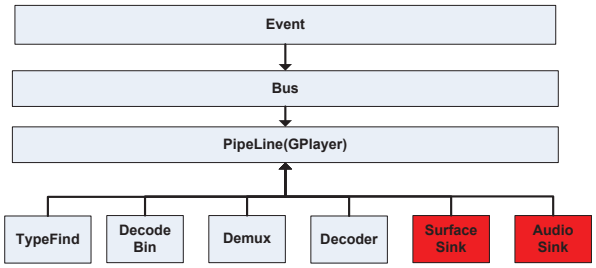

Fig. 6. The whole system after extension

First, because Gstreamer is a plug-in based multimedia engine, the whole system needs to load many different functional dynamic libraries to coordinate to complete the whole video processing steps. Large numbers of dynamic libraries' loading and unloading can consume much memory, hence it seriously cuts down the execution efficiency. To solve this problem, we use the Prelink technology to give every dynamic library a fixed memory loading address to reduce the execution consumption. Second, due to that different kinds of multimedia terminal devices have different CPU architectures, we must make our extension approach work well in the common CPU architectures such as ARM, X86, MIPS and SH4. Here, we modify the assembly languages in Liboil and Gstreamer engine according to different CPU architectures. Third, to deal with more video/audio containers, we add many related Gstreamer plug-ins, such as TS Demux plug-ins. Further, we fix bugs in those plug-ins when running in embedded environment to obtain a high compatibility. Last, we add the OpenMax IL standard support in Android.

5 Experiment Results and Analysis

5.1 Experiment Setup

In our experiment, we consider several different kinds of video container formats. For each video container format, we take several different video clips with different rates, decoding standards and definitions. Though both our Android OS and Gstreamer have been ported to X86, MIPS, ARM and SH4, in this experiment section, we take S3C6410 ARM processor as an example. The development board is Real6410 as show in Figure 7. And the specific parameters are: ARM11 Samsung S2C6410 up to 667MHz, 256MB mobile DDR RAM up to 266MHz, 1GB NAND Flash, Linux 2.6.28. During the experiments, we add our Gstreamer module to Android 2.1, mount the modified Android file system to development board, then install the original Java multimedia application (without any extension) released by Google to test our work.

Fig. 7. Real6410 development board

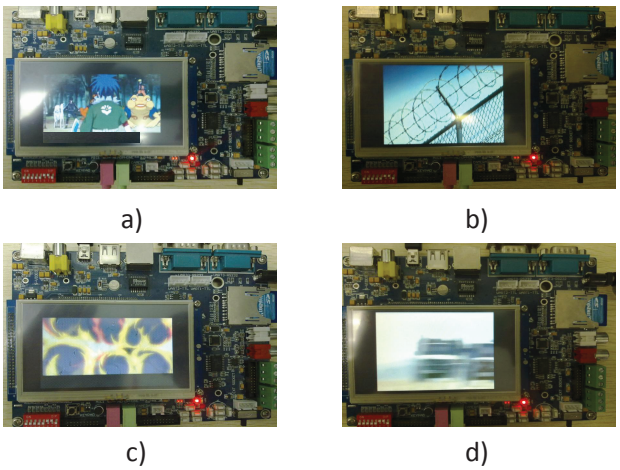

Fig. 8. Test results using AVI (a), RMVB (b), TS (c) and FLV (d)

5.2 Experiment Results

The running results of dealing with four common video container formats (i.e., AVI, RMVB, TS, and FLV) with our Gstreamer-based multimedia framework are shown in Figure 8. Detailed performance evaluations are presented in Table 1. In the column of Table 1, 1 standards for the method using OpenCore only, 2 represents the extended OpenCore in [6], 3 means the Java+FFMpeg+NDK approach in [7] [8], and 4 refers to our new Gstreamer based extension approach.

From Figure 8 and Table 1, we can see that, with our approach, android can deal with more common used video container formats without any change in Java applications. As for decoding efficiency, stability and compatibility, for

		1		2		3		4	
VideoClips(Video+Audio)		Play	Fluent	Play	Fluent	Play	Fluent	Play	Fluent
TS File	MPEG2+MP2	No	Unknown	Yes	Fluent	No	Unknown	Yes	Fluent
	MPEG2+AAC	No	Unknown	No	Unknown	No	Unknown	Yes	Fluent
RMVB File	Real+Real	No	Unknown	No	Unknown	Yes	Fluent	Yes	Fluent
	WMV+AMR	No	Unknown	Yes	Fluent	Yes	Fluent	Yes	Fluent
AVI File	WMV+MP3	No	Unknown	Yes	Fluent	Yes	Fluent	Yes	Fluent
	WMV+WAV	No	Unknown	Yes	Fluent	Yes	Fluent	Yes	Fluent
FLV File	M4V+MP4	No	Unknown	Yes	Fluent	No	Unknown	Yes	Fluent
	QT+MP4	No	Unknown	No	Unknown	No	Unknown	Yes	Fluent
MP4 File	MPEG4+MP2	Yes	Very	Yes	Fluent	Yes	Fluent	Yes	Fluent
3GP File	MPEG4+MP2	Yes	Very	Yes	Fluent	Yes	Fluent	Yes	Fluent

Table 1. Evaluation results

MP4&3GP continer formats, OpenCore is more efficient than Gstreamer. As for other video formats such as ts, mpeg, etc, OpenCore cannot deal with them, Gstreamer and approach proposed in [7] (Java+NDK+FFMpeg) almost have the same efficiency. Moreover, from Table 1, we can find that using Gstreamer as multimedia engine has a high compatibility for different video formats.

Apart from that, our approach is more feasible. For example, if a new function is needed, we only need to add some existed plug-ins. We do not need to change the Java API and all Java applications can benefit from our extension. Furthermore, our new multimedia framework can work well on different CPU architectures with an appropriate execution efficiency, thus our extension can be applied to different multimedia terminal devices. Moreover, with OpenMax IL Standard, our approach can support hardware video processing acceleration. All these show that our approach can significantly extend the Android's multimedia processing ability regarding efficiency, compatibility, feasibility and universality.

6 Conclusions and Future Work

In this paper, focusing on enhancing the multimedia processing ability of android, the extension mechanisms regarding its core media engine OpenCore are discussed and analyzed. Paying particular attention to the Gstreamer based method, we optimize the commonly used Gstreamer based android multimedia frameworks and propose another android multimedia framework which owns good working efficiency, compatibility, feasibility and universality. As for our future work, we plan to consider some other factors such as using hardware Overlay system to accelerate video display output [16].

Acknowledgment

Part of this work has been supported by *Instituto de Telecomunicações*, Next Generation Networks and Applications Group (NetGNA), Portugal and by Na-

tional Funding from the FCT-*Fundação para a Ciência e a Tecnologia* through the PEst-OE/EEI/LA0008/2011 Project.

References

1. Wang, X., Vasilakos, A., Chen, M., Liu, Y., Kwon, T.: A Survey of Green Mobile Networks: Opportunities and Challenges. ACM/Springer Mobile Networks and Applications, (2011)
2. Felker, D.: Android Application Development For Dummies. For Dummies, Australia, (2010)
3. Meier, R.: Professional Android 2 Application Development. Wrox Press, USA, (2010)
4. Conder, S.: Android Wireless Application Development. Addison-Wesley Press, Boston, USA, (2010)
5. Ableson, F.: Android in Action. Greenwich: Manning Publications, UK, (2011)
6. Song, M.Q., Sun, J., Fu, X.L., Xiong, W.K.: Design and Implementation of Media Player Based on Android. In: 6th International Conference on Wireless Communications, Networking and Mobile Computing, September (2010)
7. Song, M.Q., Sun, J., Fu, X.L.: Research on Architecture of Multimedia and Its Design Based on Android. In: International Conference on Internet Technology and Applications, pp. 1-4, August (2010)
8. Fu, X.L., Wu, X.X., Song, M.Q., Chen, M.: Research on audio/video codec based on Android. In: 6th International Conference on Wireless Communications, Networking and Mobile Computing, September (2010)
9. Gaignard, B.: GStreamer as multimedia framework in Android: a new alternative. In: CELF Embedded Linux Conference Europe (CELF ELF Europe), October (2010)
10. Gstreamer [EB/OL], http://gstreamer.freedesktop.org/
11. OpenMAX Integration Layer Application Programming Interface Specification Version 1.0. The Khronos Group Inc, (2005)
12. OpenMAX Development Layer Interface Specification Version 1.0.2. The Khronos Group Inc, (2005)
13. Gstreamer TI [EB/OL], https://gstreamer.ti.com/
14. Alejandro, A.R., Mireya, S.G., Sunil, K.: Streaming media portability with the emerging support OpenMAX. IETE Technical Review (Institution of Electronics and Telecommunication Engineers, India), vol. 28, pp. 146-157 (2011)
15. Truman, T.E.: InfoPad multimedia terminal: A portable device for wireless information access. IEEE Transactions on Computers, vol. 47, pp. 1073-1087 (1998)
16. Lee, S.C., Jeon, J.W.: Evaluating performance of android platform using native C for embedded systems. In: International Conference on Control, Automation and Systems, pp. 1160-1163 (2010)