Google's Android as an application environment for DTV decoder system

Nikola Kuzmanovic
Faculty of Technical Sciences
Novi Sad, Serbia
nikola.kuzmanovic@rt-rk.com

Tomislav Maruna, Milan Savic, Goran Miljkovic,
Djordje Isailovic
RT-RK d.o.o.
Novi Sad, Serbia

Abstract—**This paper presents an approach of using Google's Android software stack as an application environment for digital television (DTV) sets and set-top boxes. Expanding Android software stack to support DTV decoder devices and developing universal TV and set-top box applications for Android environment enables rapid software development and shorter release time for next generation products. Paper describes the process of porting complete Android software stack (Android Linux kernel, system drivers, user space libraries, Android specific libraries and programs) to a DTV decoder platform with Linux kernel and available stable device drivers; implementation of extensions to support DTV channel search, memorizing, playback and recording. Performance and features of final system will be presented.**

Keywords – DTV set, set-top box, Linux kernel, embedded systems, Android software stack

I. INTRODUCTION

Today, DTV (digital television) sets and set-top boxes (in further text multimedia devices) are controlled by custom proprietary software (operating systems and applications), written generally for one specific model or for a family of similar models. Reusability of software code between different multimedia device models is very sparse, thus software development requires a solid amount of time and resources for one multimedia device model development cycle.

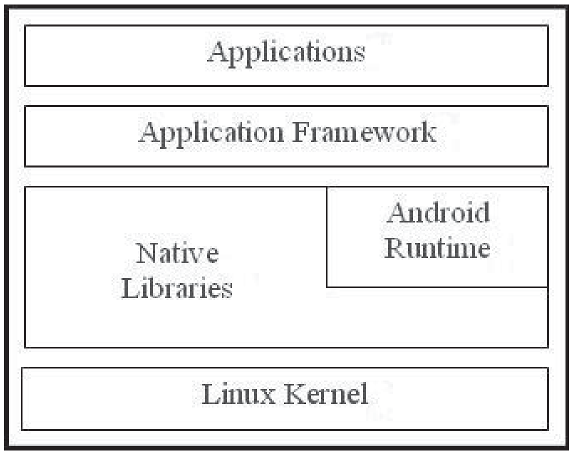
Amount of differences between hand-held and modern home multimedia devices is steadily decreasing and the similarity of used hardware is becoming considerable (processing power of CPUs, RAM size, flash memory size and availability of other integrated devices: LAN and Wi-Fi LAN, bluetooth, HW accelerated graphics, etc). Over the time primary purpose of hand-held and multimedia devices hasn't changed, although new and more advanced features have been added constantly to both fractions (video recording, internet TV, surfing the web, etc.).

Android [1] is the first open, complete and free software platform created for mobile devices and developed by The Open Handset Alliance (group of over 30 companies led by Google). Android is becoming very popular in embedded market for two mainstream reasons. First, source code is completely free; moreover there are no royalty fees for Java VM. Second, deriving from the first, Android is highly suitable for expansion and enhancement as the developer see fit.

This work was partially supported by the Ministry of Science and Technological Development of Republic Serbia under the project No. 11005, year 2008

This paper presents an approach of using Google's Android software stack as an application environment for embedded multimedia devices, mainly DTV decoder systems and set-top boxes. Motivation for porting and modification of Android rests in its availability (open source) and used Linux kernel hardware abstraction layer (also open source). Android software stack and Linux kernel enable developers to access multimedia, network connectivity, graphics, telephony, etc. This enables the process of building universal applications used on multiple Android devices, without considering the specific hardware on which the application may run.

II. ANDROID SOFTWARE STACK



Picture 1. Android System Architecture

Android relies on Linux kernel for core system services such as security, memory management, process management, networking stack and driver model. The kernel also acts as an abstraction layer between the hardware and the rest of the software stack.

Android native libraries are written in C/C++ and used by various components of Android system. The capabilities are exposed to the developers through the Android application framework. Some of the core libraries: system C library (a BSD derived implementation of a standard C system library (libc), tuned for embedded Linux-based devices), media libraries (based on PureVideo OpenCORE; libraries for playback and recording of many popular audio and video formats, as well as image files), webkit library (a well known modern web browser engine which powers the Android browser), OpenGL ES 3D library (the libraries use either hardware 3D acceleration if available or the highly optimized 3D software rasterizer implementation), FreeType (bitmap and vector font rendering), SQLite (a powerful and lightweight relational database engine available to all applications), etc.

Android run-time includes a set of core libraries that provide a substantial subset of features offered by core libraries of Java programming language. Android applications run in separate process instances utilizing Dalvik VM [2] (register-based VM, running classes compiled by Java language compiler that have bin transformed to Dalvik Executable (.dex) format optimized for embedded devices with a small memory footprint [3]). Dalvik VM uses Linux kernel for underlying functionality such as threading and low-level memory management.

Granting the full access to the framework APIs used by the core applications, developers are free to take advantage of the device hardware, location information, run background services, set alarms, add notifications, reuse and replace existing components. In a nutshell, Android provides an open development platform offering developers the ability to build modern and powerful applications.

Top layer is the Application layer. Application layer accommodates many built-in applications: SMS program, e-mail client, Calendar, Maps, Web Browser, Phone, etc., as well as custom developed applications downloaded from internet or installed via an SDcard device.

III. System Overview

For the purpose of this project Android software stack is ported to a genuine DTV decoder device (Micronas IDTV development platform [4]). Device consists of an integrated system on chip (SoC) and peripheral devices. SoC contain one 240Mhz MIPS 24kc Linux dedicated processor, multiple A/V dedicated decoder processors, graphics processing unit (GPU), graphic accelerator (GA), universal serial bus (USB) controller, etc. Available software: 2.6.21.7 Linux kernel with stable drivers (for all devices in SoC), GNU gcc/glibc cross toolchain, supported DirectFB graphic library, etc.

IV. Android Porting

Porting Android to a big-endian platform is a process divided into three stages. Used Micronas platform comes with Linux kernel version 2.6.21.7 which is pretty old for a kernel today. In spite of this, frameworks used by the platform drivers make it much harder to port them into the Android 2.6.27 or later kernel version (amount of work exceeds the purpose of this project) than to do the opposite. Consequently, in the first stage, Linux kernel drivers used by Android software stack need to be back-ported to the platform supplied kernel. The second stage of the porting process demands extension of Android build environment [5] to support big-endian MIPS platforms; the settings from the environment have to be properly interpreted in the compilation process. Final stage of porting Android software stack consists of audio driver integration and DirectFB video frame-buffer support.

A. Linux Kernel Back-porting

Few new drivers are introduced to the Linux kernel tree to support Android software stack. Some of the drivers are:

- ASHMEM (Anonymous SHared MEMory) – shared memory manager similar to POSIX SHM but with different behavior; it supports a simpler file-based API,
- Binder for Android – a CORBA like IPC (Inter Process Communication),
- Android event logger – Linux kernel driver for cataloguing debug messages directly in kernel.

Linux kernel 2.6.21.7 tree configuration that came with the used platform had to be modified in number of ways to support Android software stack and mentioned new drivers had to be integrated.

B. Android Environment Modification

Android software stack had been successfully ported to little-endian MIPS platform [9]. Due to the different endianes of the used platform (big-endian) modifications to Android software stack are numerous.

- The first kind of modifications involves platform native software binaries, mainly open-source libraries and programs written in C/C++ (endianes support is typically maintained through internal build systems (primarily “autoconf”)). Android build system makefiles (Android.mk) were changed to support missing big-endian variables and defines.
- The second kind of modifications was the build tools in-code endian unawareness. These bugs are rare and hard to find as they generally cause Android build to fail or prevent the platform from eventually starting. Programs that were affected are Android pre-linker Apriori, stripping application Soslim, resource manager aapt (Android Asset Packaging Tool), etc.
- The third and final kind of modifications includes addition of support for in-code endian awareness for libraries and programs running on Android. Central example is the Dalvik VM (that explicitly supports big-endian platforms). Uses of pointer assigning for variable data types works correctly for little-endian architectures, but makes the code not portable to big-endian platforms. Culprit was the use of a structure JValue throughout the VM code as a one structure for many value types container. Classical misuse of correct little-endian variable data types pointer assigning. Bug of same sort was found in the logger library. Few more modifications included fixing of color issues, mainly endian sort, in SKIA engine, Pixel and Surface Flinger.

C. Android DirectFB Support

DirectFB (Direct Frame Buffer) framework is a GNU/Linux/UNIX based software library with a small memory footprint that provides graphics acceleration, input device handling, etc. on top of Linux frame-buffer device. Because it is open source, various DTV SoC vendors use it to implement their frame-buffer IP-s drivers.

Android supports standard Linux frame-buffer device driver as a primary drawing surface. As DTV SoC vendors implement hardware accelerated DirectFB video drivers, support for DirectFB was implemented on various levels throughout Android software stack, firstly, to support the drawing of the UI and secondly, to accelerate it by provided graphic hardware.

- DirectFB graphic allocation device was implemented to support allocation of (double buffered) primary video frame-buffer memory, as well as all other kinds of surfaces in various color spaces.
- DirectFB copybit library was also implemented to support hardware accelerated 2D blitter. If the provided DirectFB driver supports 2D blitting operations (blit, stretch-blit, between various color spaces) central CPU unit will be pardoned from process consuming software blit tasks.

Only setback of using DirectFB as the UI graphic frame-buffer is that it doesn't natively support 3D acceleration that Android can utilize. Steady development of DirectFB library over last couple of years (there were attempts to add OpenGL like 3D acceleration) make it promising that we could expect 3D hardware acceleration possible over DirectFB library, if hardware 3D acceleration IP is present.

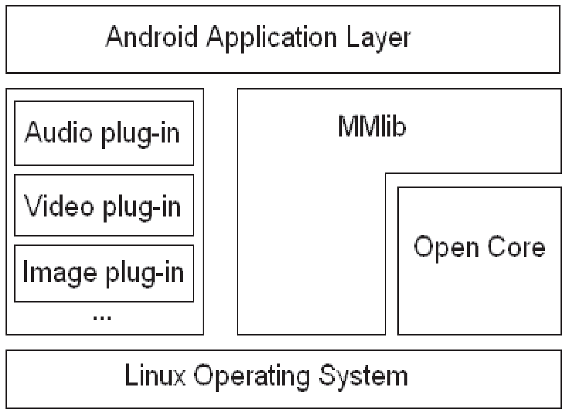
V. Multimedia and DTV Playback Support

This section of the paper explains modification and expansion of Android runtime and application framework to support DTV and multimedia device control. First step will modify Android to support hardware audio and video decoders and allow local multimedia file playback (this step is very important, as CPU-s found in DTV and multimedia decoder devices are not that powerful to do software multimedia decoding; instead they use integrated multimedia hardware accelerated devices). Second step will add support for DTV channel search and memorizing. Third and final step will add support for DTV channel playback and recording.

A. Multimedia Playback Support

Android Java runtime libraries own a media object "android.media", an interface used to control playback of audio/video files and streams. Android implementation relies heavily on an open-source software "OpenCORE" media manager library (originally contributed to Android by PacketVideo) that performs all acquiring, processing, decoding and encoding of multimedia data and more. This approach deeply relies on CPU processing power which DTV and set-top box platforms most probably don't have.

To support hardware accelerated multimedia playback, media object framework will be extended with options to allow hardware to process, decode and present multimedia data and to notify Android (state changes) only through events. This will be done with the addition of hardware plug-in mechanism (further called MMlib).



Picture 2. MMLib library Architecture

MMlib library acts as the glue layer between hardware accelerated platform decoders and Android media object. It is situated in the "android_media_MediaPlayer" JNI object. Initialization of MMlib interface queries installed hardware platform plug-ins for hardware support information and stores appropriate player interfaces (plug-ins). Using the provided support information, when media player object sends multimedia playback command, MMlib interpreter processes the command and chooses the best playback interface for the completion of the request. If no hardware acceleration is supported MMlib passes the command to the OpenCORE library (preserving the default procedure defined in Android).

Micronas platform supports various video and audio container formats. Therefore, for the purpose of this project, support for these audio, video and image container formats is implemented through several separate plug-ins.

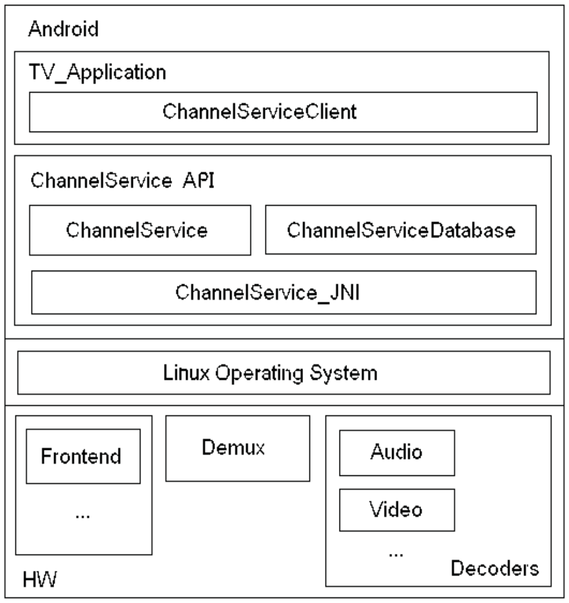
B. DTV Channel Search and Memorizing

DTV sets and set-top boxes commonly include various frontend devices to provide analog and digital channel reproduction. To enable channel search and memorizing, in Adroid programming manner, we introduce two Java classes (one process service that utilizes the objects), one user interface and one JNI interface object to support channel service natively.

Introduced objects are:

- ChannelService thread

Channel service thread represents the background process that utilizes the channel service JNI interface. Primary function of this service is to collect and modify information about supported platform frontend devices as well as capabilities these devices possess. Secondary function of this service is to asynchronously provide support for manual and automatic channel search as well as support for collection of various kinds of channel services offered by broadcasted data in memorized channels and to support plug-ins that provide this information over the internet. It stores and uses information from the ChannelServiceDatabase object it maintains.



Picture 3. Channel Service Architecture

- ChannelServiceDatabase object

Channel service database should provide a highly versatile way to store information about all various channel types (DVB-T, DVB-S, DVB-C, Analogue, Radio, etc. and all their subtypes and types to come). With the help of SQL database, Android provides a very good framework for flexible storing of data. The ChannelServiceDatabase object is not directly visible to the end user. Its contents can be browsed and modified through ClannelServiceClient interface.

Implementation of channel service database provides only channel search support and memorizing but not overall Android awareness. Awareness support is achieved by synchronization of the channel database with local android file system, by providing files with channel description. These files have a known mime type. These files are parsed as new multimedia objects and they represent instructions for the Android multimedia framework.

- ChannelServiceClient interface

Channel service client interface presents a gateway to the Channel service subsystem. It implements a number of helper functions and objects used commonly with frontend, channel and channel service management while controlling the way user or users utilize them. Because Adroid can have more then one TV application running at one time ChannelServiceClient interface has to implement asynchronous access to ChannelService subsystem.

To use the channel service subsystem Android application only needs to implement this interface.

- Channel service JNI interface

This interface presents the link between native platform frontend framework and implemented Java channel service. It provides functions for querying platform capabilities, and mechanisms for obtaining frontend, channel and channel service information. The interface compiles and builds as a shared object. It is dynamically opened from Java program code, to check if the support for channel service exists on the platform.

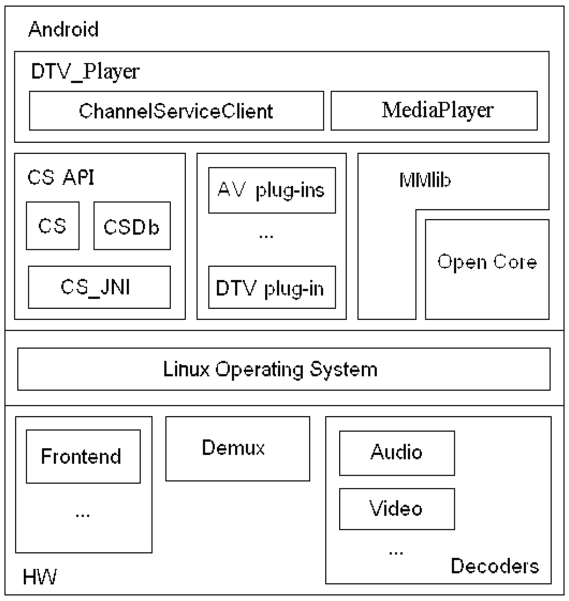
C. DTV Channel Playback and Recording

By expanding the capabilities of media object framework to support MMlib library and by defining the channel service multimedia files, DTV device control and playback implementation becomes just an MMlib plug-in.

Implementation of the DTV playback and recording MMlib plug-in is platform dependent. On Micronas platform DTV interface is implemented directly by kernel drivers, so the implementation of the DTV plug-in was straight forward, as the instructions of channel settings were received from the channel service multimedia files.

VI. ANDROID TV APPLICATION

Android platform doesn't have a DTV playback application. However, from previous passages of this document, because of modifications and additions to the Android software stack, DTV playback is natively supported by Android.



Picture 4. DTV Player Application Overview

For the purpose of testing all DTV platform capabilities DTV Player application was implemented.

VII. TESTING RESULTS

Android platform was designed for mobile devices and is still a software platform under extensive development.

However, when ported properly on a new system it automatically becomes capable of executing built-in Android applications. Although conceived for hand-held devices with small screens, ported Android operates acceptably on tested DTV decoder device. The GUI speed is seldom sluggish, provided that the tested platform has a rather slow CPU. This poses no problems for recent set-top boxes that have as much as twice the power as used Micronas platform.

Modifications and extensions made on Android software stack, to support DTV channel search and playback, were tested with the DTV Player application. Tests included channel search, memorizing, playback and recording.

Compared to the platforms documented in [6] [7] [8] and present commercial solutions, where multimedia devices have finite specified number of options, Android powered platforms give the user ability to add options, modify the experience, in other words, change the way they use embedded multimedia devices entirely.

VIII. Conclusion

Android, today, is not intended for use on a DTV platform. However, with modifications and extensions it could become a fully capable software environment for new embedded multimedia devices. As an open-source platform, Android can bring nearly infinite amount of possibilities through expansions and core modifications and new application development.

This paper presents a new way for multimedia device software development. By enabling DTV services for Android, on Micronas IDTV platform, together with built-in Android applications (web-browser, java games, picture, audio and video managers, etc.), TV set utilization experience has been changed completely. Use of Android software stack (or some future Android derivative) in embedded multimedia systems could become a reality in the near future, enhancing the overall experience expected from modern multimedia systems.

References

[1] Google Android, http://www.android.com

[2] Dalvik Virtual Machine, http://www.dalvikvm.com/

[3] Analysis of Dalvik Virtual Machine and Class Path Library, http://imsciences.edu.pk/serg/wp-content/uploads/2009/07/Analysis-of-Dalvik-VM.pdf

[4] Micronas IDTV development platform, http://www.micronas.com/pressroom/press_releases/articles/0812/index.html?newslang=1

[5] MIPS Android, http://www.mips.com/android

[6] Lalit Kumar, Rajesh Kushwaha, Rishi Prakash: "Design & Development of Small Linux Operating Systems for Browser Based Digital Set Top Box", 2009 First International Conference on Computational Intelligence, Communication Systems and Networks, pp. 277-281

[7] Qiong Li, Minqiang Guo: "Digital Recordable Integrated Television Based on Embedded Linux Operating System", 2009 World Congress on Computer Science and Information Engineering, pp. 81-84

[8] Ganesh Sivaraman, Pablo Cesar, Petri Vuorimaa: "System Software for Digital Television Applications", 2001 IEEE International Conference on Multimedia and Expo (ICME'01), pp. 154

[9] MIPS Android source code, http://www.linuxfordevices.com /c/a/News/MIPS-to-release-Android-source-code-by-August