

Guilherme de Maio Nogueira

***GingaNCL for Android: Lua code execution with
graphical support***

Vitória - ES, Brasil

February, 2011

Guilherme de Maio Nogueira

***GingaNCL for Android: Lua code execution with
graphical support***

Monograph presented as partial requirement for
obtaining the B.Sc. degree on Computer Sci-
ence at the Federal University of Espírito Santo
(UFES)

Advisor:
Prof. Dr. Magnos Martinello

DEPARTAMENTO DE INFORMÁTICA
CENTRO TECNOLÓGICO
UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO

Vitória - ES, Brasil

February, 2011

Monograph of the Final Project entitled “*GingaNCL for Android: Lua code execution with graphical support*”, defended by Guilherme de Maio Nogueira and aproved in February, 2011, at Vitória, state of Espírito Santo, by the examination of the following comission:

Prof. Dr. Magnos Martinello
Advisor

Prof. Dr. Roberta Lima Gomes
Co-Advisor

Prof. Dr. José Gonçalves Pereira Filho

B.sc. Giovanni Ventrone Comarela

Acknowledgements

This work was developed at LPRM-UFES (Laboratory of Research in Networks and Multimedia) as part of the *GingaFrRvo* and GingaRAP projects in collaboration with the Telemidia laboratory of PUC-RIO. It was partially financed by *Rede Nacional de Pesquisa* (RNP) through its Center of Research and Development of Digital Technologies for Information and Communication (CTIC) and by CAPES - Brazil (RH-TVD #225/2008).

Thanks to

Well, I believe I should thank everyone that helped me. But naming those would be too hard. So I would like to express to everyone that has helped and accompanied me during these last 6 years, during these 24 years of living, my greatest appreciation.

I would also want to express my gratitude towards all the great musicians out there. Without music life would be extremely boring and pointless.

"There is no such thing as a good influence, Mr. Gray. All influence is immoral—immoral from the scientific point of view."

"Why?"

"Because to influence a person is to give him one's own soul. He does not think his natural thoughts, or burn with this natural passions. His virtues are not real to him. His sins, if there are such things as sins, are borrowed. He becomes an echo of some one else's music, an actor of a part that has not been written for him. The aim of life is self-development. To realize one's nature perfectly—that is what each of us is here for. People are afraid of themselves, nowadays. They have forgotten the highest of all duties, the duty that one owes to one's self. Of course, they are charitable. They feed the hungry and clothe the beggar. But their own souls starve, and are naked. Courage has gone out of our race. Perhaps we never really had it. The terror of society, which is the basis of morals, the terror of God, which is the secret of religion—these are the two things that govern us. And yet—" [...]

"And yet," continued Lord Henry, in his low, musical voice, and with that graceful wave of the hand that was always so characteristic of him, and that he had even in his Eton days, "I believe that if one man were to live his life fully and completely, were to give form to every feeling, expression to every thought, reality to every dream—I believe that the world would gain such a fresh impulse of joy that we would forget all the maladies of mediaevalism, and return to the Hellenic ideal—to something finer, richer than the Hellenic ideal, it may be. But the bravest man amongst us is afraid of himself. The mutilation of the savage has its tragic survival in the self-denial that mars our lives. We are punished for our refusals. Every impulse that we strive to strangle broods in the mind and poisons us. The body sins once, and has done with its sin, for action is a mode of purification. Nothing remains then but the recollection of a pleasure, or the luxury of a regret. The only way to get rid of a temptation is to yield to it. Resist it, and your soul grows sick with longing for the things it has forbidden to itself, with desire for what its monstrous laws have made monstrous and unlawful. It has been said that the great events of the world take place in the brain. It is in the brain, and in the brain only, that the great sins of the world take place also"

Abstract

Brazil is currently ongoing the process of implementing its new digital television standard, expecting to fully replace analog transmissions by 2017. This standard, referred to as SBTVD, provides support for transmission to portable devices, including support for executing interactive applications using the Ginga middleware.

Although smartphone sales in the country have increased and the government has required that a percentage of manufactured devices has to have SBTVD support, only a few devices with this feature are available in the market today.

This work aims to improve the implementation of Ginga-NCL for Android based devices, developed at UFES, in order to provide a Lua execution machine, which is required by the standard. And, with this, to provide an open-source implementation for this kind of devices that can give an overview of the features of this middleware.

Resumo

O Brasil está atualmente no meio do processo de adoção do seu novo padrão de televisão digital, com previsão para substituir todas as transmissões analógicas até o ano de 2017. Este padrão, conhecido como SBTVD, contempla suporte à transmissão para dispositivos portáteis, incluindo o suporte a execução de aplicações interativas usando o middleware Ginga.

Apesar das vendas de smartphones no país terem aumentado, e a existência de uma portaria interministerial requerendo que uma porcentagem dos dispositivos comercializados no país tenham suporte ao SBTVD, apenas alguns poucos aparelhos com esta característica estão disponíveis hoje no mercado.

Este trabalho consiste em melhorar a implementação do Ginga-NCL para dispositivos baseados no sistema Android, desenvolvida pela UFES, de modo a prover uma máquina de execução Lua, como requerido pelas normas do padrão. Objetivando, com isto, fornecer uma implementação de código aberto para este tipo de dispositivos que seja capaz de demonstrar as características deste middleware.

Contents

1	Introduction	p. 12
1.1	Objective	p. 13
1.2	Organization of This Work	p. 13
2	The Brazilian Digital Television System	p. 14
2.1	A Brief Historical Description	p. 14
2.2	Defined Standards	p. 15
2.2.1	Transmission	p. 15
2.2.2	Coding	p. 15
2.2.3	Multiplexing	p. 17
2.2.4	Receivers	p. 17
2.2.5	Security	p. 18
2.2.6	Interactivity Channel	p. 18
2.2.7	Operational Guide	p. 20
2.2.8	Middleware	p. 20
2.3	The Ginga Middleware	p. 20
2.3.1	Ginga Overview	p. 21
2.3.2	Ginga versus other digital television <i>middleware</i>	p. 24
3	Ginga and the Android Platform	p. 26

3.1	The Android Platform	p. 26
3.1.1	What is Android Made of	p. 28
3.1.2	Dalvik Virtual Machine	p. 29
3.1.3	SDK and NDK	p. 30
3.2	Ginga-NCL for Android	p. 31
4	Lua support for Ginga on Android	p. 36
4.1	The Lua Interpreter	p. 37
4.1.1	Preliminary test	p. 37
4.2	Lua Modules in SBTVD	p. 38
4.3	Lua Canvas	p. 38
4.3.1	Validation Tests	p. 43
5	Conclusion and Future Work	p. 45
	Bibliography	p. 46
	Annex	p. 48
	Annex A - Source Code of the Test Application	p. 48
	Annex B - Source Code of the Lua Related Classes	p. 49

List of Figures

2.1	Transmission system block diagram	p. 15
2.2	Different divisions of a single channel	p. 16
2.3	Bidirectional interactivity channel recommended architecture	p. 20
2.4	SBTVD standards and Ginga (SOARES; BARBOSA, 2008)	p. 21
2.5	Ginga Components	p. 22
2.6	Ginga <i>Common Core</i> Components	p. 23
3.1	Smartphone Operating System U.S. Market Share - May 2010	p. 27
3.2	Android System Overview	p. 29
3.3	Architecture of Ginga-NCL for portable devices	p. 32
3.4	Formula 1	p. 33
4.1	AndroidLuaCanvas class diagram	p. 39
4.2	AndroidLuaSurface class diagram	p. 41
4.3	AndroidLuaPlayer class diagram	p. 42
4.4	Graphical elements drawn from Lua	p. 44

List of Tables

2.1	Video coding standard of SBTVD	p. 17
2.2	Audio coding standard of SBTVD	p. 17
2.3	One-seg Receiver Features	p. 19
2.4	Main digital television <i>middleware</i> comparison	p. 24
3.1	Smartphone Operating Systems Global Market Share, Q3 2010, by Gartner, Inc.	p. 28

Introduction

In recent years, two relatively new technologies have gained much attention in Brazil: digital television and smartphones. The Brazilian Digital Television system was standardized three years ago and is considered to be one of the most advanced digital television standards in the world. Its implementation promises a high degree of digital inclusion in the country, as census show that over 90% of Brazilian families have a television at home. Based in this potential to digital inclusion, the Presidential Act 5820 (BRASIL, 2006), established the rules for the implementation of SBTVD. It states that by 2013 the digital television signal must cover the entire country territory, and that by 2017 all television transmission should be digital. By that time, all analog channels concessions should be returned to the State.

These digital inclusion is linked to the execution of interactive applications transmitted together with the television signal. The format of these applications is defined by the standard, and they should be executed in the multitude of devices capable of receiving the DTV signal. To standardize the application execution, a middleware is specified, named Ginga middleware. Ginga was developed specially to meet the Brazilian needs and, besides being an advanced digital television middleware, it is also recommended by ITU as middleware for IPTV services (ITU-T Recommendation H.761, 2009).

SBTVD standards include support for transmission and interactive applications execution for portable devices, intersecting with the fast growing market of smartphones in Brazil. The growth of smartphones sales, allied with the arrival of newer operating systems for these devices, allowed an environment where developers can have applications easily built and delivered to millions of consumers. With hardware getting more powerful and cheaper, devices capable of receiving the digital television signal have appeared in the market, and the government, foreseeing this smartphone application trend stated, through (BRASIL, 2008), that by January of 2010 the cellphone device manufactures should add Ginga-NCL to at least 5% of their total device production. This statement caused a lot of discussions and was later changed in 2009 to extend the date to July of 2011 and change the percentage to 1% of the production.

In the academic sphere, there were also initiatives to study combination of these two areas, such as

the implementation of Ginga-NCL for Symbian based devices, developed at PUC-RIO (SOARES, 2008) and, more importantly, the basis of the current work, the implementation of Ginga-NCL for Android based devices, developed at UFES. The later was part of the *GingaFrRvo* and GingaRAP projects of LPRM-UFES (Laboratory of Research in Networks and Multimedia) and the Telemidia laboratory of PUC-RIO, and was financed by RNP/CTIC.

1.1 Objective

One of the future works cited in (DAHER et al., 2010) is adding support for the Lua execution environment to the implementation. This is one of the major works that have to be done, since Lua support is mandatory for Ginga-NCL, including portable devices.

The first challenge to add support for Lua execution in the Ginga implementation based on Android devices is how to embed the Lua interpreter, which is usually implemented in C, given that Android development relies on Java language. The second challenge is related to the characteristics of interactive applications for digital television that rely heavily in graphics, thus, they need the Lua Canvas support for allowing the manipulation of graphic primitives and images. In this case, the requirement is to design and to be able to control directly the drawing functions in the Android platform.

The main objective of this work is to provide a Lua execution environment to the previous implemented Ginga-NCL middleware for Android based devices, focusing on the canvas library: the Lua graphics manipulation library. In order to validate the current Lua support implementation, some NCL applications were developed having Lua code as media objects. These applications allow to explore some resources supported by Lua module e.g. to present conditional graphical objects, working as a proof of the concept.

1.2 Organization of This Work

The work begins by summarizing the Brazilian Digital Television System, on Chapter 2, with special attention to the Ginga middleware. Chapter 3 then follows with an overview of the Android platform and the description of the previous implemented Ginga-NCL for this platform. After that, Chapter 4 presents the work on adding support for Lua to the implementation referred in the later chapter, describing how the Android API's were used to achieve the features defined by the standard. Finally, Chapter 5 presents final considerations and future work.

Chapter 2

The Brazilian Digital Television System

This chapter presents the technologies used within the Brazilian Digital Television System, known as SBTVD, focusing on the standardized middleware Ginga and the associated languages.

2.1 A Brief Historical Description

Discussions on the digital television matter have started a long time ago in Brazil, in 1994, with a partnership between the Brazilian Society of Television and Engineering (SET) and the Brazilian Association of Radio and Television Broadcasters (Abert), as stated in (MONTEZ CARLOS; BECKER, 2004), which formed a research group to study digital television adoption in the country.

In 1998, the National Agency of Telecommunications (Anatel) took over the leadership of this research, aiming at choosing an already established international standard for adoption. The technical discussion of these standards ended on 31 August 2000, concluding that the Japanese standard, ISDB, was the most appropriated for the country. A public announcement of the chosen standard was expected to follow, but was postponed to wait for the next government, that would start in 2002.

After the new government office took place, the Ministry of Communications, Miro Teixeira, sent a letter of intentions to the President, where he expressed his concerns about digital inclusion through interactive TV. Following this letter, an announcement was made stating that the country would develop its own digital television standard, and a work group was formed to study the matter again. With the Presidential Act number 4.901 in November of 2003, the Brazilian Digital Television System (SBTVD) was instituted and three committees were created: Development Committee, Consultant Committee and the Management Group (BRASIL, 2003). In 2006 the final decision was announced with Presidential Act number 5.820 (BRASIL, 2006), stating that Brazil adopted the Japanese standard as a baseline for ISDB-Tb, the commercial name for the Brazilian standard.

2.2 Defined Standards

The standards of SBTVD are defined in a set of 19 documents, grouped in eight major subjects: Transmission, Coding, Multiplexing, Receivers, Security, Interactivity Channel, Operational Guide and Middleware.

2.2.1 Transmission

(ABNT NBR 15601, 2008) defines transmission standards for SBTVD. In special it defines the concept of segments, which is useful for the transmission to portable receivers. As it is stated, during transmission, one or more MPEG-2 transport streams are (re-)multiplexed to create a single TS, that will be subjected to channel-coding steps in order to generate the OFDM (Orthogonal Frequency Division Multiplexing) signal. Figure 2.1 shows a diagram of this process.

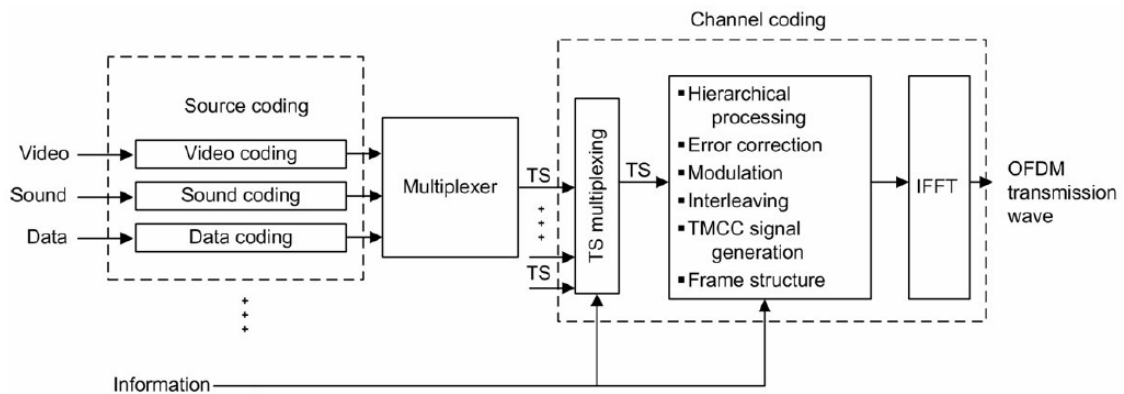


Figure 2.1: Transmission system block diagram

It specifies that the digital transmission spectrum broadcasting is to consist of 13 successive OFDM blocks/segments. With this, a part of a single television channel can be used for fixed-reception terminals and the rest for portable-reception terminals. The central segment may be submitted to frequency interleaving without the participation of other portions of the spectrum, thus allowing the creation of portable services by the reception of a single layer of the television service. This is the reason portable receivers are also called *one-seg* receivers.

Figure 2.2 show different ways a single channel transmission spectrum can be divided, in order to accommodate different services.

2.2.2 Coding

Coding is defined in (ABNT NBR 15602, 2008), and is split in three parts. Part 1 specifies video coding. A digital video signal has two types of data redundancy: *spatial* (or *intra-frame*) and *temporal*

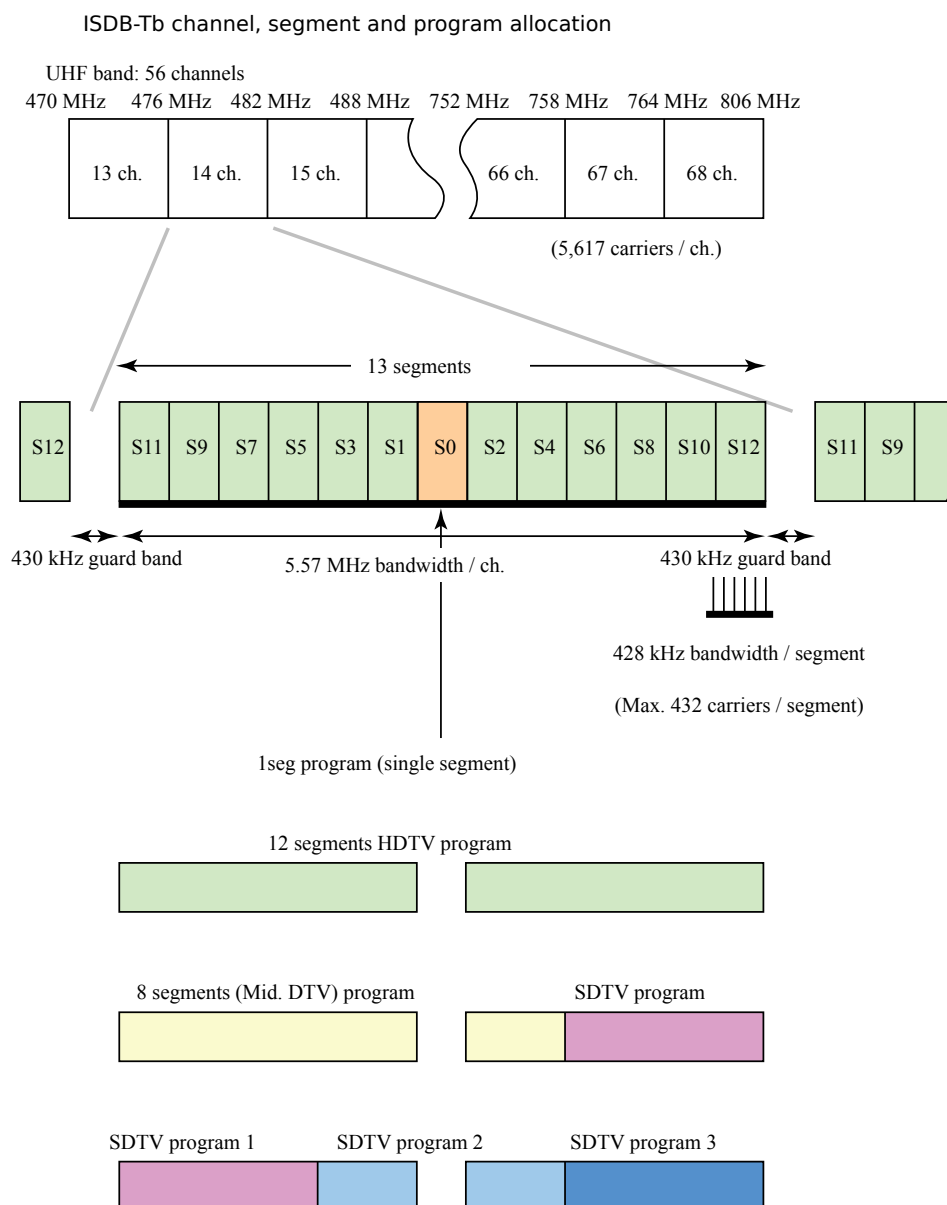


Figure 2.2: Different divisions of a single channel

(or *inter-frame*). SBTVD uses a recent codification technique that considers both types of redundancy: the H.264 (ISO/IEC 14496-10, 2004), also known as MPEG-4 Part 10 or MPEG-4 AVC (*Advanced Video Coding*). In order to best suit different classes of receivers, this standard is composed of several profiles and levels. High profiles are used for fixed and mobile receivers, while base profiles are used for portable receivers. A brief summary of this is presented in Table 2.1.

Part 2 specifies the audio coding, using the MPEG4 AAC codec (ISO/IEC 14496-3, 2004) and defining sample frequency values and quantization. Table 2.2 shows the main characteristics of this audio coding standard for each class of receiver devices.

Part 3 specifies the signal multiplexing systems, stating how audio and video should be encapsulated in PES (Packetized Elementary Stream) using the MPEG2-TS format and talks briefly about the PSI

Table 2.1: Video coding standard of SBTVD

	Fixed and Mobile Receivers	Portable Receivers (one-seg)
Standard	ITU-T H.264 (MPEG-4 AVC)	ITU-T H.264 (MPEG-4 AVC)
Level@Profile	HP@L4.0	BP@L1.3
Display resolution	480 (4:3 or 16:9)	SQVGA (160x120 or 160x90)
	720 (16:9)	QVGA (320x240 or 320x180)
	1080 (16:9)	CIF (352x288)
Frame rate	30 and 60Hz	15 and 30Hz

Table 2.2: Audio coding standard of SBTVD

	Fixed and Mobile Receivers	Portable Receivers (one-seg)
Standard	ISO/IEC 14496-3 (MPEG-4 AAC)	ISO/IEC 14496-3 (MPEG-4 AAC)
Level@Profile	AAC@L4 (multichannel 5.1) HE-AAC v1@L4 (stereo)	HE-AAC v2@L3 (two channels)
Sampling rate	48KHz	48KHz

tables.

2.2.3 Multiplexing

(ABNT NBR 15603, 2008) defines the Multiplexing process of SBTVD. Its main focus is the description of the PSI (Program Stream Information) Tables, stating the relationship between different tables and how the contents of these these tables should be stored, transmitted and parsed. The relevance of this information to the current work is minimum, specially taking into account the lack of a tuner module in the current implementation. However, this is a brief description of the content of this set of documents:

Part 1 names all the PSI tables, providing a small description of each, lists their packet identifiers (PIDs) and table identifiers (TIDs) and presents diagrams of their data structures.

Part 2 provides detailed description of each table item, including possible values and details of the transmission repetition rate.

Part 3 shows the syntax and usage of extended information of SI (Service Information).

2.2.4 Receivers

(ABNT NBR 15604, 2008) defines how receiver should be built. It specifies what features are mandatory and optional for full-seg and one-seg receiver. Among its information are:

- The definition of the basic architecture of an IRD (Integrated Receiver Decoder) device;
- Environment and safety conditions, such as temperature and humidity;
- Specification of the antenna capacity, channel reception, channel bandwidth, frequency and sensitivity;
- Accessibility resources;
- and others

In particular importance for this work is the Annex B, which contains information on what features of the Ginga middleware are mandatory when implemented for one-seg devices. Table 2.3 shows a short list of these features.

2.2.5 Security

The security features presented in (ABNT NBR 15605, 2008) are related to copy protection. The protection of the digital broadcast content is to be defined by rules in the transport stream coding that are reflected in the application running at the receivers, in order to attain to this rules and replicate them in recorded signals.

However, one-seg receivers are not required to implement copy control mechanisms and the one-seg transport stream should be signaled as "copy-free".

2.2.6 Interactivity Channel

The protocols and physical interfaces used to provide interactivity channel for SBTVD are described in (ABNT NBR 15607, 2008). Perhaps the most important information to this work is the basic architecture for bidirectional interactivity channel based on TCP/IP protocols and the definition of interface types for mobile phone connection.

The basic architecture is shown in Figure 2.3. It demonstrates how the middleware can communicate with interactive application servers and positions the interactivity channel outside of the main broadcast system. As for interface types, the standard acknowledges mobile phone connection technologies such as EDGE, GPRS and EVDO and specifies the *MOBILE_PHONE* type, which represents a mobile phone connection without an specified technology. Since 3G networks (HSDPA) and 4G are newer standards, they would fall in this category.

Table 2.3: One-seg Receiver Features

Static formats (monomedia)		
Image	PNG with restrictions	Mandatory
	PNG without restrictions	Optional
	GIF	Optional
	MPEG-2 "I - Frame"	Optional
	MPEG-4 "I - VOP"	Optional
	H.264 / MPEG4 AVC "I - Picture"	Mandatory
	JPEG with restrictions	Mandatory
	JPEG without restrictions	Optional
	MNG	Optional
Audio	MPEG-2 Audio AAC	Optional
	MPEG-4 Audio AAC-LC	Mandatory
	MPEG-1 audio (Layer 1 and 2)	Mandatory
	MPEG-1 audio layer 3 (MP3)	Optional
Video	MPEG-2 Video "drips"	Optional
	MPEG-4 Video "clips"	Optional
	H.264 / MPEG-4 AVC "clips"	Optional
Broadcasting Formats		
Program video		Mandatory
Program audio		Mandatory
Subtitles		Optional
Closed Caption		Optional
Libras		Optional
Fonts		
Resident	Tiresias	Optional
	Verdana	Mandatory
Downloadable	All types	Optional
Broadcasting Channel Protocol		
IP Multicast	MPEG-2 Section Filter	Mandatory
	Objects carousel - DSM-CC	Mandatory
	Data carousel - DSM-CC	Optional
Interactivity Channel Protocol		
*		Optional
Program Language		
NCL		Mandatory
Java		Optional
Bridge		
LUA		Optional
ECMAScript		Optional
Execution Engine		
LUA		Mandatory
Java		Optional

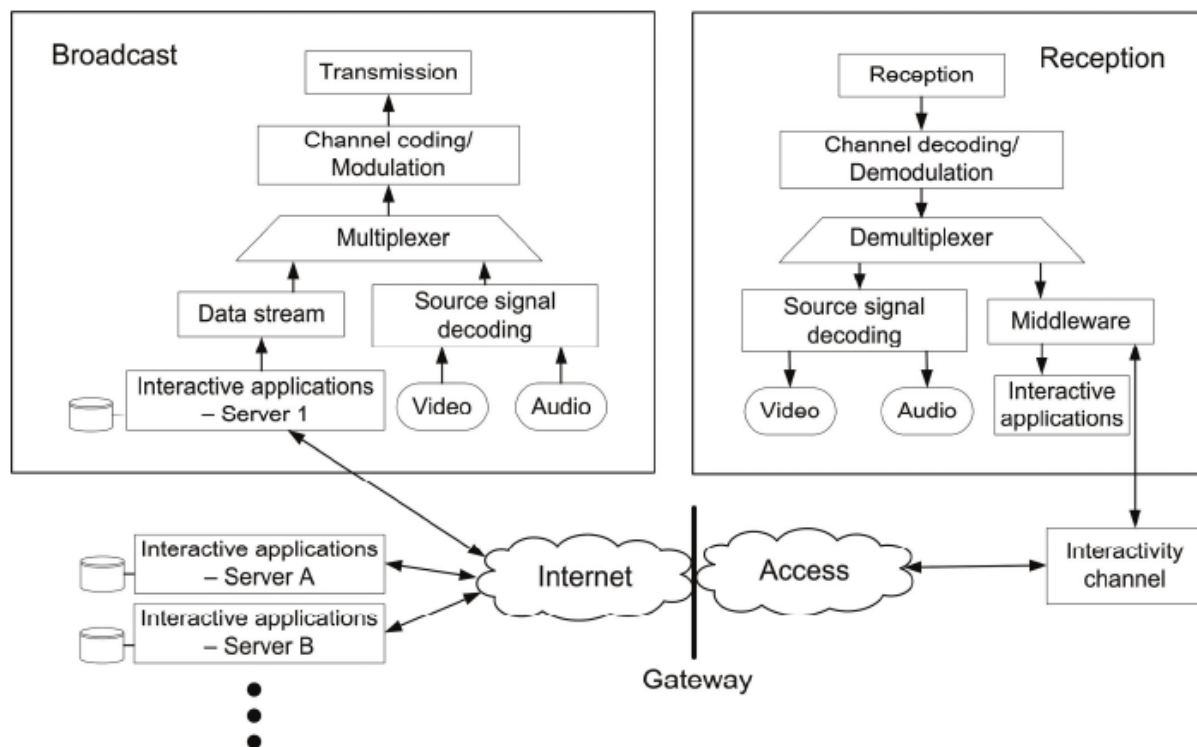


Figure 2.3: Bidirectional interactivity channel recommended architecture

2.2.7 Operational Guide

The operational guide (ABNT NBR 15608, 2008) contains guidelines for the implementation of the standards of SBTVD. Currently, their content is not in the lines of this work, since only the guidelines for transmission, multiplexing and video/audio coding have been released.

2.2.8 Middleware

Finally, (ABNT NBR 15606, 2008) defines the middleware Ginga, the object of this monograph. Given its importance, this subject is described in the next Section.

2.3 The Ginga Middleware

Until not a long time ago, in computing systems, programs were written and compiled for specific platforms (operating system and hardware) each time. In recent years many solutions have arisen to overcome this paradigm. A classic, and successful, example is the Java programming language. Java programs are compiled into a bytecode that is run inside a virtual machine that is standardized, however, there are implementations of this JVM (Java Virtual Machine) for several different platforms. In this way

a Java program can be written and compiled one time and run in different platforms, making it highly *Portable*.

Portability is a fundamental requirement for digital television applications. There are a myriad of set-top-boxes and televisions from different manufacturers, and each one of these devices can have a specific hardware, operating system and embedded libraries, and an interactive application transmitted within the digital television signal must be executed in equal form in all these devices.

It would be extremely hard to application developers to deal with this problem in the application side. Therefore, to circumvent this problem, digital television standards usually specify a *middleware* to be used. In this context, a middleware is an intermediate software layer that lays on top of the operating system and provides the functionality to execute the transmitted applications. Each digital television standard specifies its own middleware that, in general, is developed according to the specific need of the country or region where this standard will be adopted.

This section aims to present Ginga, the middleware specified by the Brazilian Digital Television System, and to compare it with middleware of different international standards.

2.3.1 Ginga Overview

As discussed previously, Ginga is the codename of the Brazilian Digital Television middleware, and its main functionality is to provide platform abstraction to the interactive applications, in order to execute in different devices in the same way. Figure 2.4 shows how Ginga is positioned in a layer view of SBTVD.

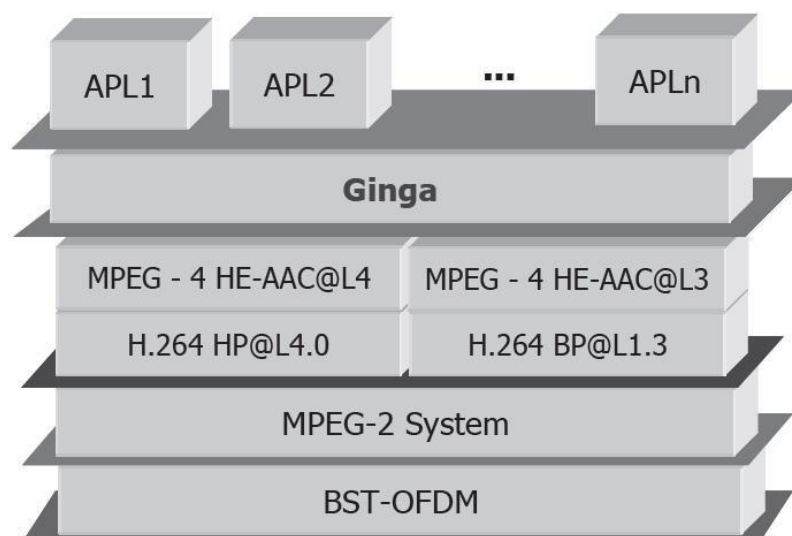


Figure 2.4: SBTVD standards and Ginga (SOARES; BARBOSA, 2008)

Architecture

The Ginga architecture was designed according to the needs of digital television applications. In general, applications can be developed following either one of these paradigms: procedural or declarative. Declarative applications are better suited for simple applications that rely simply on media synchronization, while procedural applications can be more complex. To deal with these two possibilities, Ginga has two subsystems: Ginga-NCL (SOARES; RODRIGUES; MORENO, 2007), the declarative subsystem, and Ginga-J (FILHO; LEITE; BATISTA, 2007), the procedural one.

These subsystems share a third architectural component, the *Ginga Common Core*. Figure 2.5 presents a top-level view of the Ginga components, and the relation between them. Although Ginga-J (execution engine) and Ginga-NCL (presentation engine) are separated components, the Bridge provides communication, in order that declarative application can refer to procedural applications and vice versa.

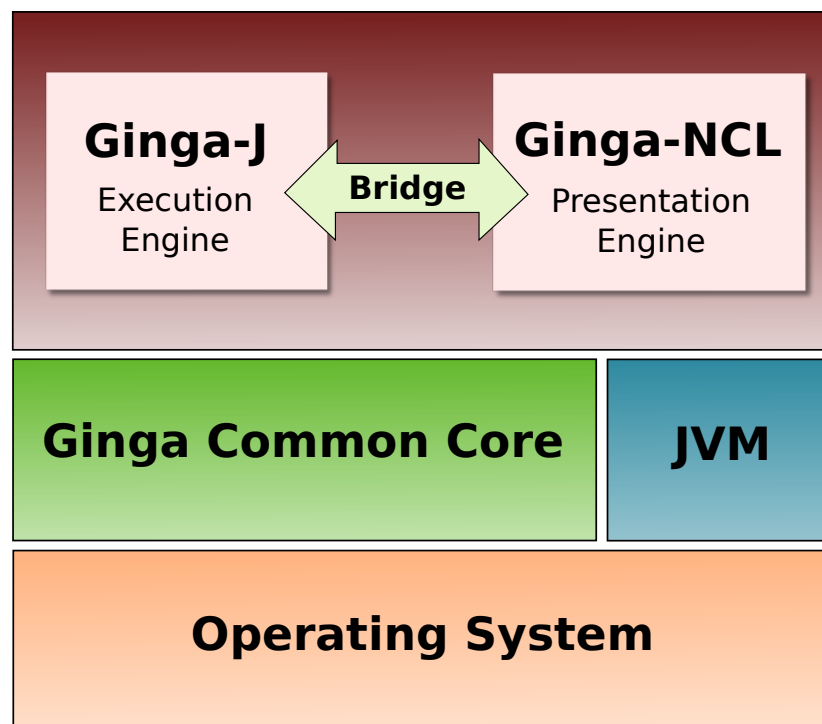
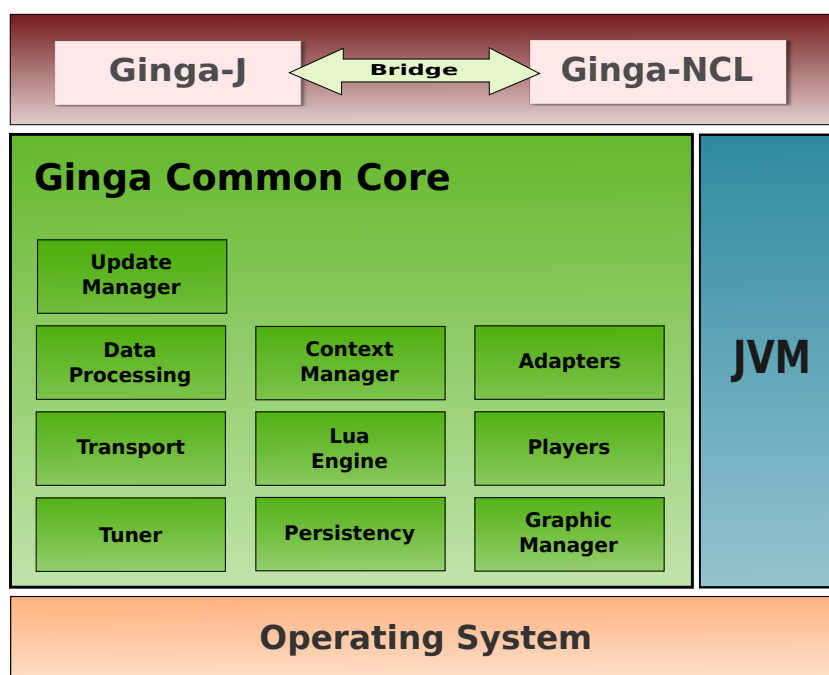


Figure 2.5: Ginga Components

Ginga Common Core

Ginga Common Core provides services necessary to the proper functioning of both the declarative and procedural environments. Among these services we can cite the decoding and demultiplexing of the MPEG-2 transport stream, graphic output control, user input managements, etc. Figure 2.6 presents a view of the components of Ginga Common Core, where we can identify the services cited, among others.

Figure 2.6: Ginga *Common Core* Components

Ginga-NCL

Ginga-NCL is the subsystem responsible for the declarative environment of Ginga. Its main functionality is the processing of NCL documents, which is an XML application language with features to provide easy specification of interactivity, space-time synchronization between media objects, adaptability, support for multiple exhibition devices and support to live creation of non-linear interactive applications.

Ginga-NCL provides API's to process XHTML document including CSS and ECMAScript, and an LUA execution machine. The Lua execution machine is one of the main differences of Ginga-NCL to other used middleware. Lua is a scripting language heavily used and very lightweight. Its execution machine, which is register-based, is one of the fastest for interpreted languages, being often used as benchmark for other interpreted languages. The Lua environment has modules specifically designed for SBTVD, and are discussed in Chapter 4.

It is also important to note that, for one-seg devices, Ginga-NCL is the only mandatory subsystem of Ginga (Table 2.3).

Ginga-J

Ginga-J is the subsystem responsible for the processing of DTV applications written in Java language, also known as *Xlets*. While adding new features, it also maintains backwards compatibility with most of the digital television middleware adopting the GEM standard (GEM is an unified specification

for digital television middleware proposed by the European DVB group (ETSI TS 102 819, 2005)).

Some important requirements for the Brazilian digital television context were not satisfied by the GEM international standard. In order to satisfy these specific requirements and still maintain compatibility with GEM's APIs, Ginga-J was defined using three set of APIs as base:

- *Green*: is the set of all APIs compatible with GEM. It is composed of the JavaTV (Sun Microsystems, 2008), DAViC (Davic, 1998), HAVi (HAVi, 2001) and DVB (ETSI TS 102 812, 2003) standards;
- *Yellow*: are extensions proposed to satisfy the Brazilian specific requirements. This set is not compatible with GEM, but can be implemented in software using the Green APIs.
- *Blue*: this set is composed of integration APIs. They provide mean for an STB to communicate with another device that provides a compatible interface. These extensions are very innovative and are not compatible with GEM.

2.3.2 Ginga versus other digital television *middleware*

Table 2.4 shows some *middleware* and their main features regarding both declarative and procedural environments

Table 2.4: Main digital television *middleware* comparison

<i>Middleware</i>	DTV System	Declarative Environment	Procedural Environment
ACAP	ATSC N. American	ACAP-X Languages: XHTML-like & ECMAScript	ACAP-J Language: Java
MHP	DVB-T European	DVB-HTML Declarative Language: XHTML-like & ECMAScript	MHP Language: Java
ARIB-BML	ISDB-T Japanese	ARIB-BML Languages: XHTML-like & ECMAScript	Optional (GEM like) Not implemented
Ginga	SBTVD Brazilian	Ginga-NCL Languages: NCL & Lua	Ginga-J Language: Java

For the declarative environment, Ginga-NCL catches our attention as it sits aside of the XHTML with ECMAScript combination. This particularity is due to two main reasons:

- according to (SOARES; RODRIGUES; MORENO, 2007), Lua is a much more efficient language than the current implementation of ECMAScript, making it better suitable to development of interactive applications, specially games;

- while XHTML allows content structuring in many ways, their time synchronization has to be done using a script language. NCL, on the other hand, allows space and time synchronization, being a more robust declarative language. NCL does not fully replace all of XHTML features. To overcome this, Ginga-NCL has a XHTML interpreter, offering the possibility to include XHTML objects in NCL documents.

The procedural environment Ginga-J, as with the other ones, uses Java as non-declarative language. However, the Brazilian middleware exposes more advanced features to developers through the Yellow and Blue APIs, not compatible with GEM. These APIs make possible not only the interaction of the user with his television, but also user to user interaction, with support to multiple device displaying.

Without doubts, SBTVD can be considered one of the most advanced digital television systems in the world. Using recent audio and video codification, a well established modulation technique and a middleware specially created to suit the Brazilian reality, it offers more advanced features than the main international rivals.

Chapter 3

Ginga and the Android Platform

3.1 The Android Platform

Android is an open-source software stack platform targeted at mobile devices, mainly at smartphones and tablets. It was originally developed by Android Inc., a company purchased by Google in 2005. Since then, Google has worked on it together with the *Open Handset Alliance*, a business alliance of over seventy companies working to develop open standard for mobile devices.

The *Open Handset Alliance* announced Android on November 5, 2007 (Open Handset Alliance, 2007). Parts of the announcement are quoted below:

With nearly 3 billion users worldwide, the mobile phone has become the most personal and ubiquitous communications device. However, the lack of a collaborative effort has made it a challenge for developers, wireless operators and handset manufacturers to respond as quickly as possible to the ever-changing needs of savvy mobile consumers. Through Android, developers, wireless operators and handset manufacturers will be better positioned to bring to market innovative new products faster and at a much lower cost. The end result will be an unprecedented mobile platform that will enable wireless operators and manufacturers to give their customers better, more personal and more flexible mobile experiences.

Thirty-four companies have formed the Open Handset Alliance, which aims to develop technologies that will significantly lower the cost of developing and distributing mobile devices and services. The Android platform is the first step in this direction — a fully integrated mobile "software stack" that consists of an operating system, middleware, user-friendly interface and applications. Consumers should expect the first phones based on Android to be available in the second half of 2008.

The Android platform will be made available under one of the most progressive, developer-

friendly open-source licenses, which gives mobile operators and device manufacturers significant freedom and flexibility to design products. Next week the Alliance will release an early access software development kit to provide developers with the tools necessary to create innovative and compelling applications for the platform.

Android holds the promise of unprecedented benefits for consumers, developers and manufacturers of mobile services and devices. Handset manufacturers and wireless operators will be free to customize Android in order to bring to market innovative new products faster and at a much lower cost. Developers will have complete access to handset capabilities and tools that will enable them to build more compelling and user-friendly services, bringing the Internet developer model to the mobile space. And consumers worldwide will have access to less expensive mobile devices that feature more compelling services, rich Internet applications and easier-to-use interfaces — ultimately creating a superior mobile experience.

Quickly after the announcement, an early build of Android's *Software Development Kit* (SDK) was made available through Android Developers website and an application building contest was set up. During 2008, the first phone having Android was released, T-Mobile G1, developed by HTC Corp, and the 1.0 SDK version was released.

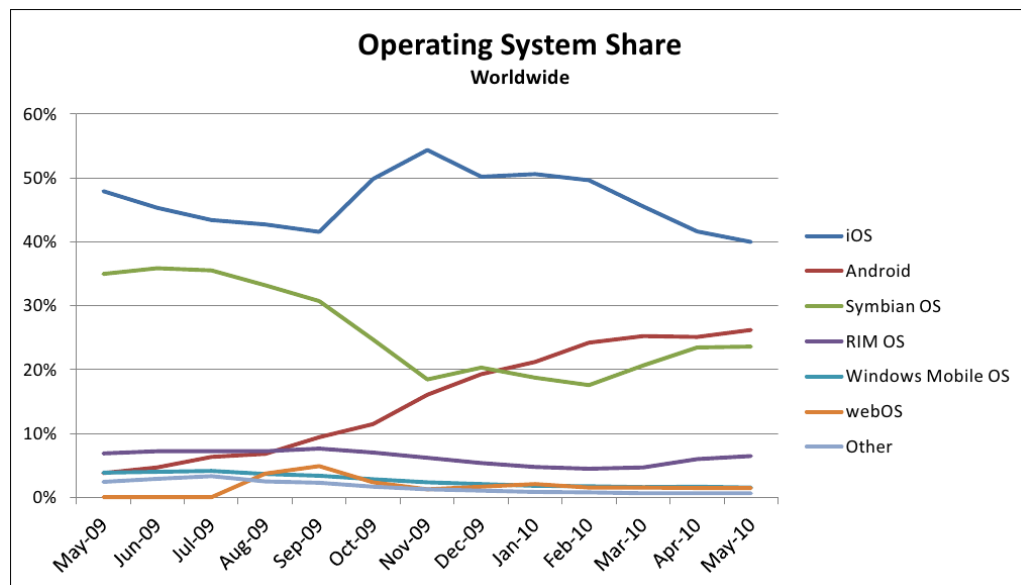


Figure 3.1: Smartphone Operating System U.S. Market Share - May 2010

In the following two years, Android gained significantly status and experienced a huge market share growth, having almost 30% of U.S market share by May 2010, according to Mobile Metrics Report (AdMob, 2010), which is presented in Figure 3.1. By the end of the third quarter of 2010, Gartner analysts released their findings on the mobile phone market (Gartner, Inc., 2010), showing growth both in overall smartphone sales and Android smartphone sales (Table 3.1).

Table 3.1: Smartphone Operating Systems Global Market Share, Q3 2010, by Gartner, Inc.

Operating System	3Q 2010 Market Share		3Q 2009 Market Share	
	Units	%	Units	%
Symbian	29,480.1	36.6	18,314.8	44.6
Android	20,500.0	25.5	1,424.5	3.5
iOS	13,484.4	16.7	7,040.4	17.1
Windows Mobile	2,247.9	2.8	3,259.9	7.9
Other OSes	14,820.2	18.4	11,053.7	26.9
Total	80,532.6	100.0	41,093.3	100.0

Currently Android is in its seventh version, 2.3 codename Gingerbread, released on 6 December 2010. The next version, codename Honeycomb, is scheduled to be released in 2011 and its main focus is providing features to better suit tablets.

This fast evolution shows the commitment of the Open Handset Alliance in pushing Android to suit up-to-date devices and technologies and the growing market share shows the large user acceptance of the platform.

3.1.1 What is Android Made of

The Android platform consists of an operating system, libraries, an application framework and a development kit. Figure 3.2 provides a high level overview of a typical Android system, including the applications. From bottom-up, it mainly consists of ¹:

1. **Linux:** Underneath everything, lies a modified version of the Linux kernel (version 2.35 on Gingerbread). Although being a Linux kernel, Android cannot be seen as a Linux distro, since it lacks many of the common libraries (such as Xlib), applications and programming frameworks of the system. At best, it could be seen as an embeddable Linux platform.
2. **System Libraries:** atop the Linux kernel, common C/C++ libraries are installed and are both used by Android components and exposed to the user via the application framework. Among these libraries are:
 - **libc:** Google has implemented a custom libc specially designed for Android, called Bionic. It has a BSD license and is optimized for being fast, as to save battery power, and small, being only 200kb, half the size of GNU libc. It has an stripped down implementation of pthread that strongly relies on the kernel.

¹The description here is based on the official Android Developers website, Tim Bray's description of Android and the two year experience as a developer.

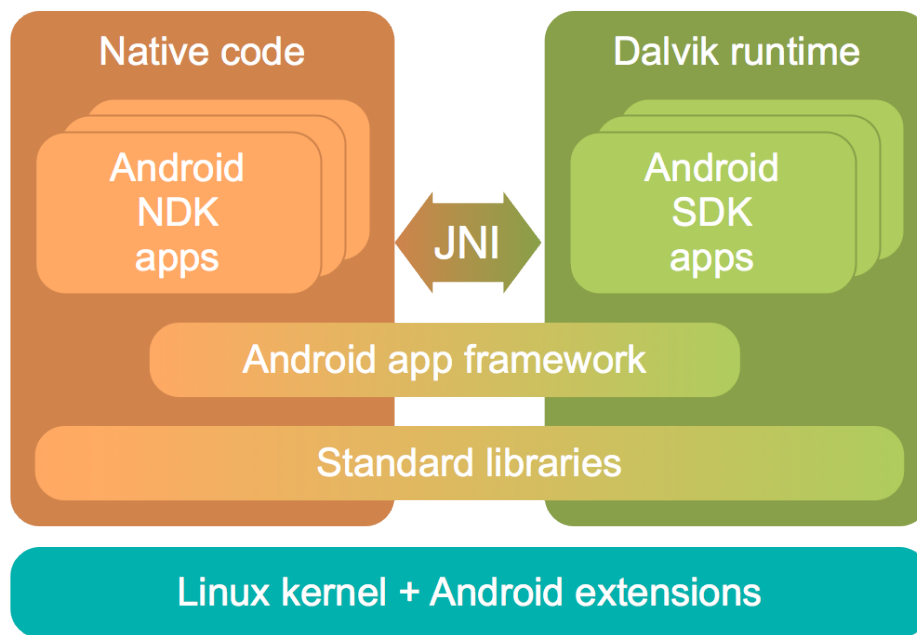


Figure 3.2: Android System Overview

- Media libraries: based on OpenCORE, they provide support for playback of video and static image formats, including MPEG4, H.264, MP3, AAC, JPG and PNG.
 - Surface Manager: a system to manage the display subsystem and compose graphic layers of the multiple applications.
 - LibWebCore: a modern browser engine, powering the Android browser and the WebView.
 - FreeType: bitmap and vector font render.
 - SQLite: the well established, lightweight relational database engine is available to all applications.
 - 3D libraries: and implementation based on OpenGL ES APIs, using 3D acceleration when available or an optimized 3D software rasterizer.
3. **Dalvik**: is the name of Android's Java Virtual Machine, and the associated runtime essential libraries and executables. It is designed specially for Android and is quite different from other Java VMs.

3.1.2 Dalvik Virtual Machine

Dalvik, as said before, is quite different from other Java Virtual Machines. In special, it has a register-based architecture, while common Java VMs are stack machines. In stack machines, instructions are used to load data into the VM stack and to manipulate that data. With register-based architecture, this is not needed, but the VM has to encode source and destination registers and generally is larger than

stack machines. However, stack machines require more instructions to implement the same code. The discussion on each architecture is the best is ongoing, but Dalvik so far has proved to be very good for mobile devices.

Another difference is the compilation to bytecode. With other VMs, Java classes are compiled into *.class* files, one for each class. With Dalvik, several *.class* files are grouped in a single Dalvik Executable (*.dex*) file, that reduces greatly the size of the application (usually an uncompressed *.dex* file is half the size of the equivalent *.class* files).

Dalvik also is not compliant with Java SE, nor Java ME profiles, since it does not implement Java ME classes, AWT and Swing. Instead it has its own library, that is built on top of a subset of the Apache Harmony Java implementation.

It is also optimized so that a device can run multiple instances of the VM efficiently, this is due to a design goal of Android that every application must run in its own instance of the Dalvik VM.

The Dalvik implementation is constantly being improved by the Android developers. Among the improvements to speed up the execution of application are: just-in-time (JIT) compilation, a technique where some parts of the code that are reused are compiled at runtime and cached for later execution (requiring more memory from the Android devices, if enabled), which was added in Android version 2.2, and the concurrent garbage collector, implemented in Android 2.3, that now runs in parallel with the other applications.

3.1.3 SDK and NDK

Developing for Android with the SDK and NDK requires an understanding of the basic components of the Android framework. Android has a different way of defining graphic elements and communication between applications, and the key components are described here.

Views

Views are the basic user interface blocks of Android, they are hierarchical and have proper methods to draw themselves. Specialized view exist for common user interface elements, such as image, buttons, text fields, etc. A developer can extend this class and override its methods to create visual elements that inherit the Views properties while better representing elements specific to the application being developed.

Activities

The concept of an activity is that of a single screen of the application, that group some kind of functionality. It can contain one or more views.

Intents

An intent is exactly what the name says, it represents the intention to perform some kind of action. Intents can be passed to an application activity, to inform an action that needs to be done. A intent can be initiated by the application or from external sources (the Android system or other applications with permission to do so), for example, to notify of an event. Intents are used to loosely couple an action and the action handler.

AndroidManifest.xml

This XML defines the properties of the application, such as its activities and permissions that it needs in order to run.

Native Environment

The native development kit was first release on July of 2009. By the time, it only provided an environment to compile shared libraries written in C or C++ to be used by Java code using the Java Native Interface (JNI), using only a few libraries part of libC. With time, this native environment got richer, with the addition of OpenGL ES, debugging capabilities with gdbserver, access to bitmaps, and others.

It was only with the release of Android 2.3, codename Gingerbread, that building an entire application in C/C++ became possible. It also added APIs to play sound from native code, using OpenSL ES, handle application events, such as life cycle, touch and key events, control windows and have access to their pixel buffer, manage EGL contexts and read assets directly of APK files. It also added a port of the C++ standard template library (STL) and support for C++ exceptions.

3.2 Ginga-NCL for Android

To implement Ginga-NCL for Android, the Java public implementation for set-top boxes was used as a base. However, a series of modifications were made, mainly in modules and methods related to graphic and audio management.

In Figure 3.3 the components that needed to be re-implemented or altered are showed with a dark background and those used without modifications are showed with a light background.

The public implementation uses graphic libraries, such as *AWT* and *Swing*, that are not supported on the Android platform. The code sections not related to this management needed little to no modification, given the portable aspect of the Java/Dalvik platform. The resulting implementation is capable of

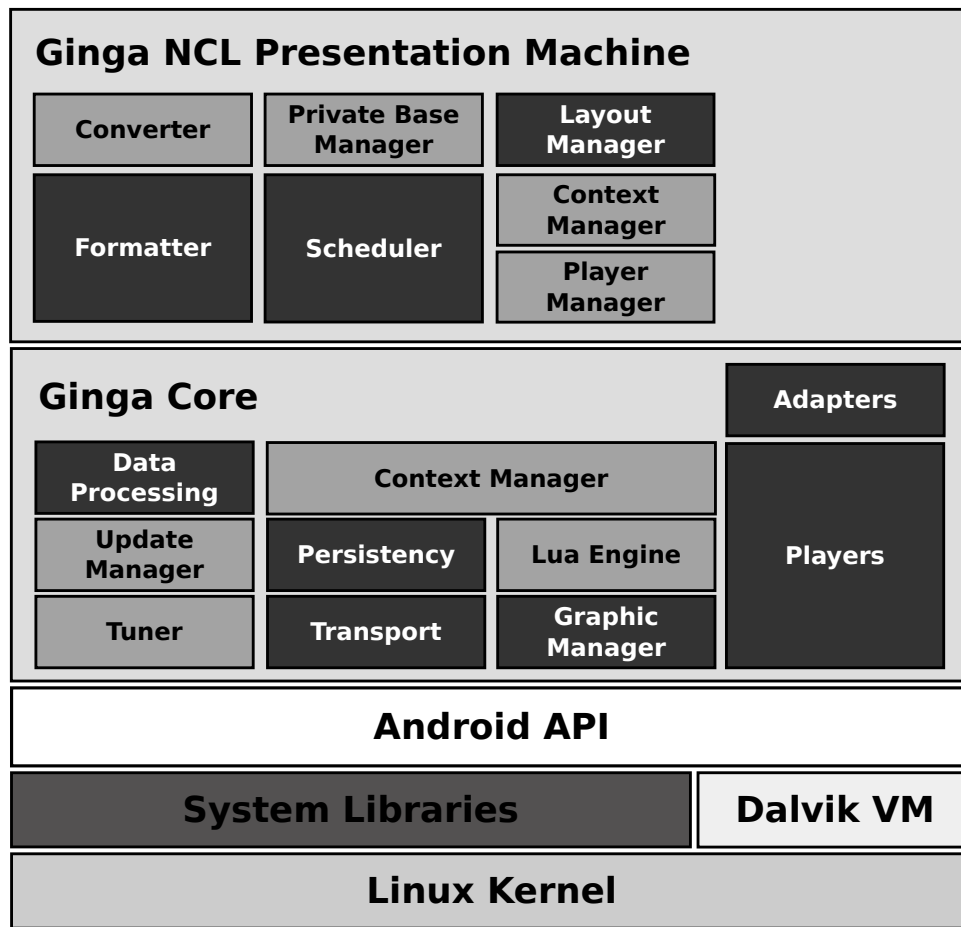


Figure 3.3: Architecture of Ginga-NCL for portable devices

executing NCL applications on portable devices satisfactorily. Figure 3.4 shows one of this applications running in an actual device with Android.

It is important to note that during the development, there were no Android based devices available in the market with an ISDB-T tuner and, as of now, the devices having this tuner do not have it exposed to developers through an API. For these reasons, the current implementation lacks a functional **Tuner** module and the **Data Processing** is in initial stage, with only a DSM-CC client implemented.

The next sections present details of the classes implemented in this first version of Ginga-NCL for Android.

Formatter

The initial instantiation of the **Formatter** class happens in the main *Activity* of the middleware, the **GingaMobile** class. The **GingaMobile** class has a reference to the **Formatter** and, after being called with a NCL application location, calls the **Formatter** to start playing it. The control buttons to pause and stop have functions calling the respective **Formatter** method after being pressed.



Figure 3.4: Formula 1

When adding an application to the Formatter, it calls the *addDocument* method of the *NclDocumentManager* class. This method will compile the NCL application into a Private Base, add this Private Base to a map and return a reference to the Formatter.

To start playing the application, the Formatter explores the Private Base structure and creates a list of entry point nodes, the initial events of the application. This list is then passed to the Scheduler, that will then initiate the execution.

Graphic Manager

The basic function of this module is to manage the addition and removal of surfaces to the screen and, therefore, it needs to alter the main layout of the application. However, the Android platform does not allow all classes of an application to alter its main layout. Only the main *Activity* has permission to do it.

To circumvent this limitation, the **InterfaceUpdater** class is implemented using a *Handler* to receive messages and run threads with privileges of the main Activity. It has two internal classes that implement the *Runnable* interface: *updateUI* e *changeImage*. Both classes have methods to create a list of elements (Windows or Views) to be either removed or added to the screen. When the thread is run, this list is consumed and the proper action is executed upon the elements.

Player Manager

In this module, the **AndroidSurface** interface may be highlighted, as it is the base interface for the display surface of every media. It is important to say that every media executed in an NCL application has an exhibition surface associated with it, and each type of media requires different surface characteristics. To attain this, the Android API provides distinct View types, such as: *ImageView*, *SurfaceView* and *WebView*.

To provide an abstraction over the Android API, the **AndroidAudioSurface**, **AndroidHTMLSurface**, **AndroidVideoSurface** and **AndroidImageSurface** classes are implemented, encapsulating the underlying Android classes.

Another important class is **AndroidWindow**. Its function is to provide an abstraction for the working window of the middleware. For it, the *LinearLayout* from the Android API is used and it implements the NCL interface **IWindow**. All elements displayed by the middleware are inside an **AndroidWindow**.

In this module there is also the **GingaKeyListener** class, that is responsible for capturing user keyboard input, and implements the *OnKeyListener* interface of the Android API.

Players

In this module the media players are defined for each media type supported by the middleware. The behavior of their methods (start, pause, abort, resume, stop, etc) are defined by the standard defined in (ABNT NBR 15606, 2008), and this implementation follows it while using the Android media API.

Transport

This initial development of the Transport and Data Processing modules implements the DSM-CC protocol (*Digital Storage Media Command and Control*) (ISO/IEC 13818-6, 2000), adopted by the SBTVD-T standard as data carousel. According to the standard, DSM-CC is used over MPEG2-TS and is transmitted multiplexed as asynchronous data packets. However, in this prototype DSM-CC was implemented over IP protocol, for its simplicity. More details on this implementation can be found in (COMARELA, 2009).

Scheduler

The NCL document scheduler is one important module of this middleware architecture, being responsible for orchestrating the execution of all elements in a given NCL application.

Its implementation has as base the reception of a NCL event, that can be instant or have a longer duration, and an action to be performed upon this event. An NCL event can be of types *Selection*, in

which an area or node of a document is selected; *Attribution*, in which a value is attributed to a property of an NCL node; *Composition*, in which a map of the composition is displayed; or *Presentation*, in which a media file contained in an NCL node is displayed. Actions over the first three types are simple and fast to execute; however, action over Presentation events are actions performed over a media file and need different treatment. This type of event has a player associated with it, that will run in a separate thread and will perform the actions over the media.

The most important action within the Scheduler is a *start* action for a Presentation event. When this happens, the player for the referenced media must be instantiated and prepared. This preparation consists of creating an exhibition surface for the media and sending it to the Layout Manager, so it is added to the current displayed surfaces list. After this preparation, the scheduler adds itself as listener to the player adapter and calls the start method of the player.

To maintain the proper synchronization of events, the Scheduler provides a callback function and register itself as listener for all non-instant events. Through this callback, the Scheduler can receive state updates for each started event and maintain an internal structure to manage conditional events. In this way, it manages the instantiation of new threads and the construction of new surfaces. The control actions for media files are issued by the scheduler to the specific players, and events generated by players, such as the end of a movie, are passed to the scheduler via the callback function.

Chapter 4

Lua support for Ginga on Android

This chapter deals with Lua support to the implemented middleware in the Android platform. It begins by giving an overview of the features of Lua language, which are the reasons behind the choice to use it as procedural language in SBTVD. Due to these characteristics, specific requirements for adding support to the language in Ginga-NCL arise and are discussed within this chapter.

Lua is a dynamically typed language intended for use as an extension or scripting language, created by PUC-RIO in 1993 (LUA.ORG). It is compact enough to fit on a variety of host platforms. It supports only a small number of atomic data structures such as boolean values, numbers (double-precision floating point by default), and strings. Typical data structures such as arrays, sets, lists, and records can be represented using Lua's single native data structure, the table, which is essentially a heterogeneous associative array.

Lua implements a small set of advanced features such as first-class functions, garbage collection, closures, proper tail calls, coercion (automatic conversion between string and number values at run time), coroutines (cooperative multitasking) and dynamic module loading.

The language has a MIT license and is highly used, specially for adding scripting capability to games¹. One of the factors that make its usage so high is its interpreter, register-based, which makes Lua one of the fastest interpreted languages available.

Given these powerful features of Lua, to be easily extensible and designed to work in conjunction with other languages, it was chosen to be the procedural language working with NCL in the Ginga middleware. In this chapter, the process of implementing the canvas module of SBTVD is described.

¹The famous MMORPG game Word Of Warcraft, for example, uses Lua for its add-ons.

4.1 The Lua Interpreter

The first step in order to add support for Lua in the Ginga implementation for Android based devices was to supply a Lua interpreter for it. The Lua virtual machine is register-based, as Dalvik, and is implemented in ANSI C, therefore, being possible to compile it to run in Android devices using Android's NDK. Since Lua is designed to be embedded in applications, it provides a C API. This C API is designed to be as simple as the Lua language itself, and is stack based. It simplifies calling Lua functions from C and C functions from Lua using the stack to pass arguments. To call a Lua function from C, you push the arguments to the stack and call it. Similarly, a C function that is to be called from Lua could extract the arguments from the Lua stack.

However, Android development is done with Java language and the Ginga implementation must integrate with the Lua C API. LuaJava is a scripting tool for Java that does this. It wraps up the Lua C API through its JNI interface and allows Lua code to manipulate Java objects and Java code to manipulate the Lua stack and implement an interface using Lua.

In order to develop a small application testing the usage of these tools together, the source code of both the interpreter ² and the C part of LuaJava were used to build a dynamic library for Android using the NDK compiler. The majority of the code was compatible with Android, but since NDK doesn't provide all of the common C libraries, some things had to be adapted. Those are the locale configuration, which provides information on the decimal numbering separator, and the console output. For the locale, the replacement behavior fixes point as decimal separator and, if the parsing fails, changes it to a comma. As for the console output, all print functions were changed to print to the Android log system, therefore being visible via the logcat tool.

4.1.1 Preliminary test

In order to validate the integration of LuaJava and LuaBinaries in an Android application, two simple tests were designed. The first one loads and executes a Lua script from the sdcard, which prints a string and the result of a sum. The excerpt of the code presented below, shows the creation of a Lua context state. The creation of this context is handled by LuaJava, which access the Lua C library and performs the initial loading. After this, the main Lua libraries are loaded into the context, in order to populate the symbols table, and resolve name references in future function calls. Finally, it loads the script from the sdcard and shows the return code of the execution in a Text View (0 for a correct execution, 1 for error). The output of the print functions used in the script is redirected to Android's global log, which is accessible through the logcat utility.

<aqui tinha um código>

²LuaBinaries

The second test application loads a Lua script from a string, and uses the Printable and ObjPrint classes to print strings from both Java code and Lua code. An excerpt is shown below, where a Lua context is created and the code in a string is loaded. Then, a function defined in the string is called, and the return is printed to Android's log system.

<aqui tinha um código>

The tests were executed both in standard way and by running the gdbserver utility, which allows remote debugging of the C library using gdb. They attested that the integration between the Lua interpreter and the Java/Dalvik environment was working in the expected way, making the implementation of the Lua modules possible.

The complete code of these test applications is available in Annex A.

4.2 Lua Modules in SBTVD

The SBTVD standard document (ABNT NBR 15606, 2008) defines the Ginga middleware in details, including the NCL language and NCLua (Lua scripts use within NCL documents). For the Lua language, it defines four modules:

- **canvas:** offers an API to draw graphical primitives and manipulate images;
- **event:** allows NCLua applications to communicate with the middleware through events (NCL and key events);
- **settings:** exports a table with variables defined by the NCL document author and reserved environment variables contained in an "application/x-ginga-settings" node;
- **persistent:** exports a table with persistent variables, which may be manipulated only by procedural objects.

Although this work focuses specially on the Canvas module, the Event module currently has a partial implementation that allows only registering an event handler, receiving and posting events, which is sufficient for starting and stopping simple applications. The other modules will be addressed in the future works.

4.3 Lua Canvas

Interactive applications for digital television rely heavily in graphics, therefore, the Lua Canvas module can be considered the most important one. This module provides a graphical API for Lua applications,

allowing the manipulation of graphic primitives and images. The functions of this module are defined in (ABNT NBR 15606, 2008), with detailed explanation of the expected behavior of each.

The Lua canvas module differs significantly from the NCL graphical environment: while in NCL you define a media file to be played in a certain region, with Lua canvas you associate a region of the screen for it to draw into. Because of that difference, they need to use the Android graphical API in different ways. NCL uses the predefined Views for exhibiting media files on the screen, and Lua extends the View class, in order to control the drawing.

In Android, to directly control the drawing, there are four required elements: a Bitmap to hold the pixel values, a Canvas to host the draw calls (modifying the Bitmap), a drawing primitive (such as a rectangle, a circle, a text or a Bitmap) and a Paint to describe the colors and styles for drawing and writing. These elements were used to develop the main class of this module, *AndroidLuaCanvas*.

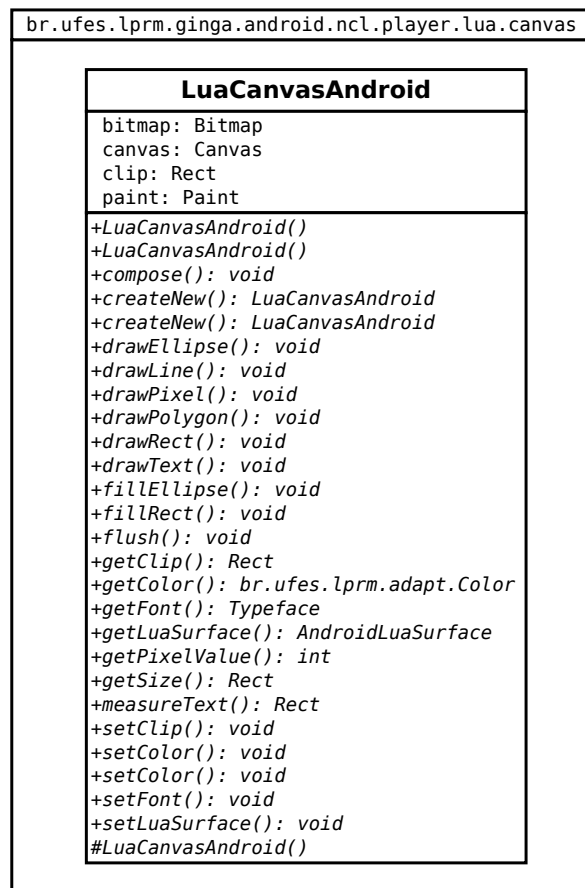


Figure 4.1: AndroidLuaCanvas class diagram

The *AndroidLuaCanvas* class implements the methods described by the standard in the following way:

- It has private objects of the classes Canvas, Bitmap and Paint, from the android.graphics package.
- There are two constructors and equivalent canvas:new() methods:

- One receives a path to an image file, open the image and creates a mutable Bitmap object containing it. The Canvas is then created using this Bitmap.
- The second, receives the desired width and height of the canvas, creates a mutable Bitmap object with these dimensions, and creates the Canvas object using this Bitmap.
- Methods that operate on attributes of the Canvas, such as setting the color for future drawing calls, and setting the font size, operate on the Paint object of the class. This object holds all the configuration for drawing in the current canvas. Additional classes act as helpers to group return values to Lua.
- Methods that perform drawing operations, operate on the Canvas object, using the associated Paint object. The drawings done through methods the Canvas object are automatically applied to the Bitmap.
- Compose operations are done by drawing the Bitmap of the source canvas onto the Bitmap of the destination canvas, at the specified position. This is done using the drawBitmap method of the Canvas, and is automatically applied to the Bitmap.
- The flush operation, which is supposed to update the canvas, does not have to operate on all canvas. The *AndroidLuaCanvas* objects act like off-screen buffers, and the drawing operations are applied without the need to flush. Therefore, the flush operation only works for the root canvas, copying it to the actual displayed element, which is an *AndroidLuaSurface* object.

The *AndroidLuaSurface* represents what is actually seen on the screen, and is directly related to the *region* tag of the NCL document. It is part of the middleware's Graphic Manager and, as with the other surfaces, is passed to the InterfaceUpdater so it can be properly displayed (Section 3.2). It implements the *AndroidSurface* interface, which in turn extends the *ISurface* interface of Ginga, and extends the *Android View* class. By extending the *View* class, it is possible to control the drawing operations on the *View*. This is needed to avoid concurrent access to the *View*'s buffer.

The *AndroidLuaSurface* overrides the *onDraw*, which receives a Canvas object as argument. This method is called by the Android graphic system every time it needs to redraw the *View*. The class has a private Bitmap object and when the middleware needs to update the contents of the *AndroidLuaSurface* class (in the flush operation of a *AndroidLuaCanvas*; Figure 4.1), it first sets the Bitmap to the one of the associated *AndroidLuaCanvas* and then calls the *invalidate* method of *AndroidLuaSurface*, which informs the graphic system that the view is no longer valid and needs to be redraw. The overridden method then receives the Canvas and draws the Bitmap onto it.

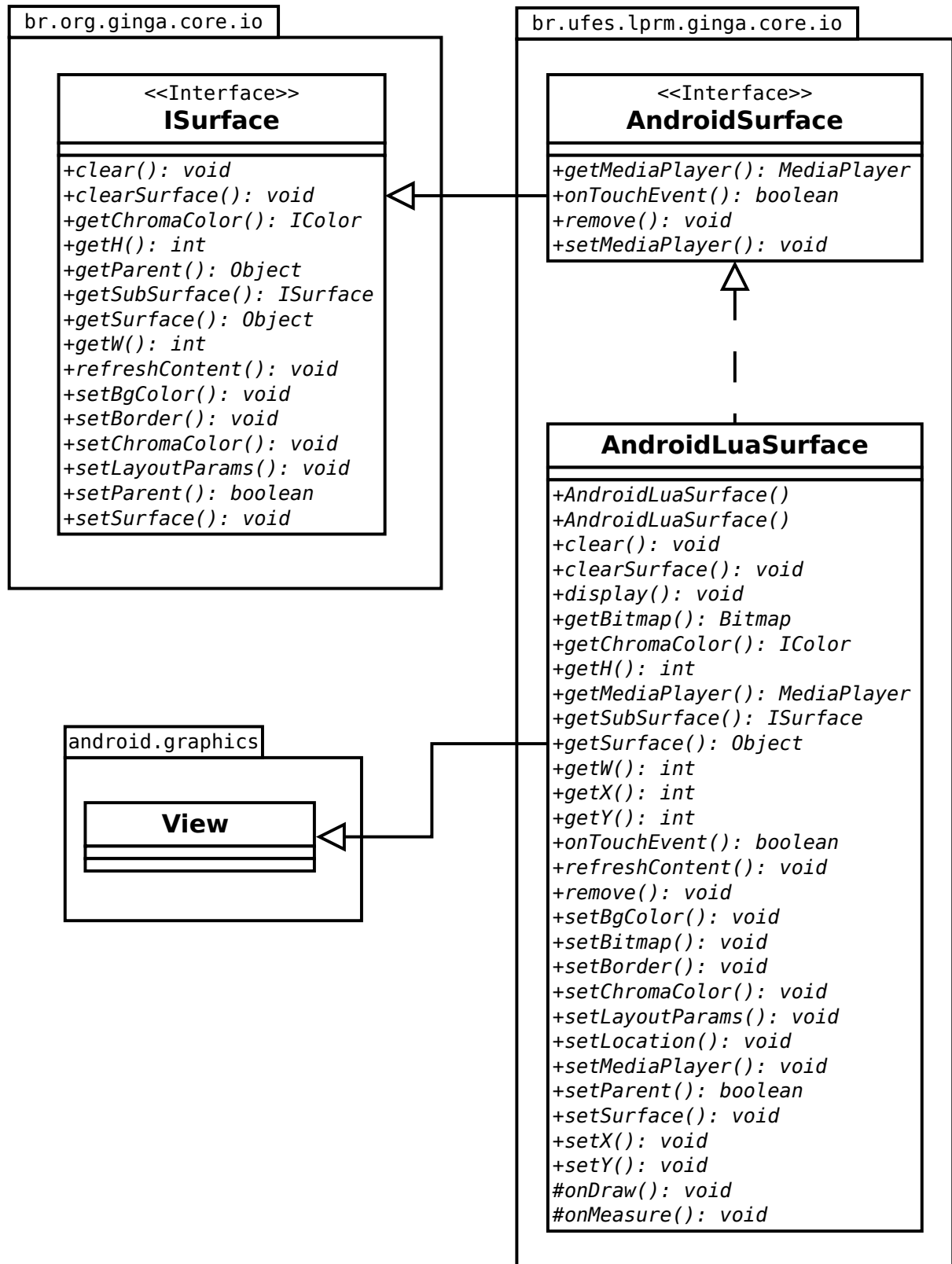


Figure 4.2: AndroidLuaSurface class diagram

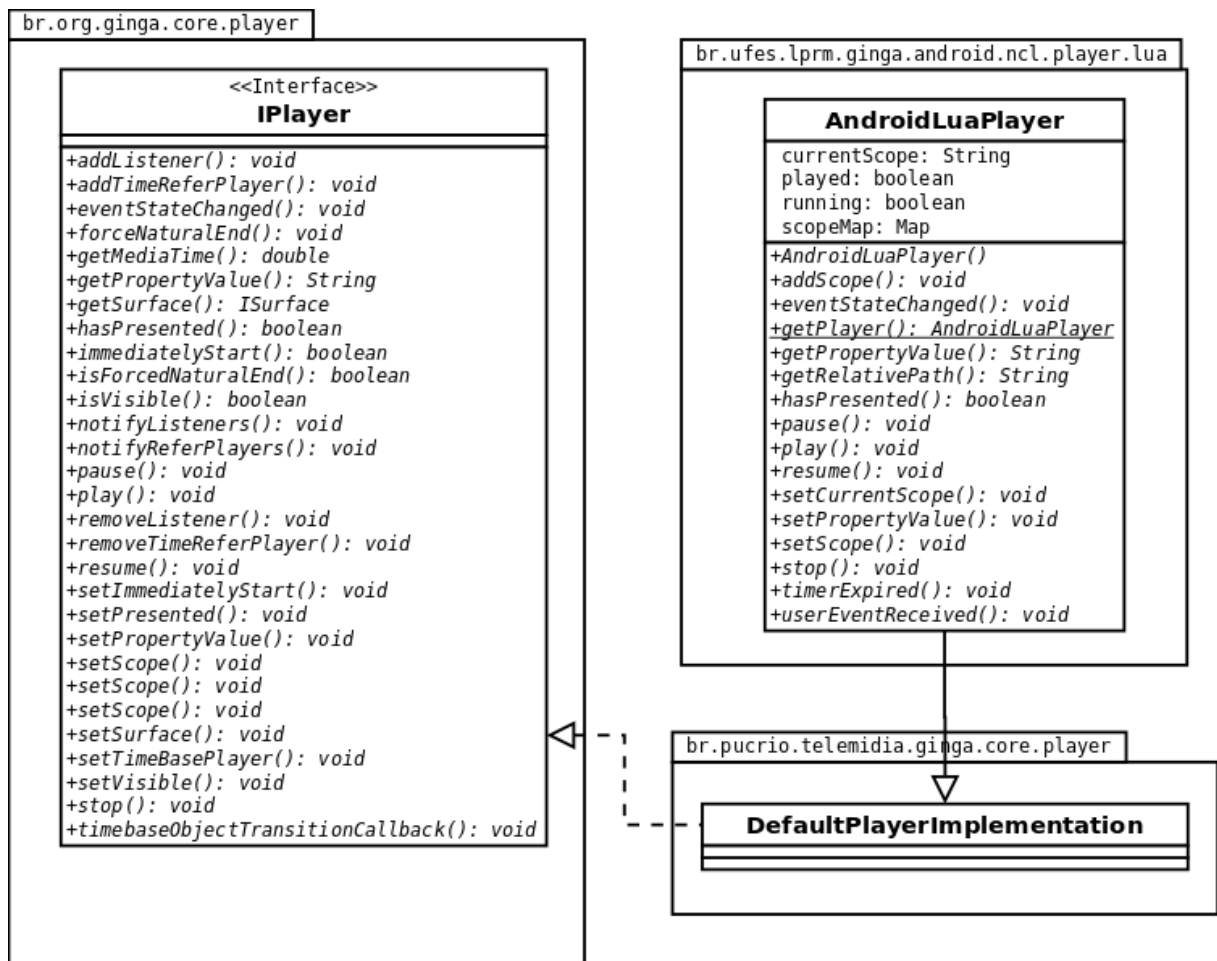


Figure 4.3: AndroidLuaPlayer class diagram

Joining these two classes together is the *AndroidLuaPlayer* class, which is responsible for loading the *canvas.lua* module (Annex B), and the Lua script defined in the NCL document. The *AndroidLuaPlayer* constructor is called in *AndroidLuaPlayerAdapter* which is used by the Scheduler to access the player when the Presentation start event is issued. It receives as argument the size of the NCL region associated to the Lua script and creates both the *AndroidLuaSurface* and *AndroidLuaCanvas* elements of this size. It then calls the *setLuaSurface* method of the *AndroidLuaCanvas* class, so that the canvas can be drawn onto the surface in the flush operation. It then sets the canvas as the main canvas in the Lua stack, and loads the Lua script.

To summarize, the execution of a NCL document containing a NCLua script follows these steps:

1. The NCL document is parsed and a private base is created as result;
2. This private base is navigated in order to get the entry points, which are passed to the Scheduler;
3. In a NCL document containing a Lua script in a media node as entry point, the Scheduler will operate the Start action over this node.

4. The Scheduler acquires a reference to the represented NCL *region*, and the AndroidLuaPlayer-Adapter. With this two objects at hand, it calls the *prepare* method of the adapter, passing the region properties, and path to script as argument.
5. The Adapter will then create the desirable AndroidLuaPlayer object, passing the width and height and path to the script.
6. During the construction of the AndroidLuaPlayer object, it will set the private object holding the path to the script and, using the width and height passed, will create a AndroidLuaCanvas object and a AndroidLuaSurface object. The Surface is associated to the Canvas by calling the *setLuaSurface* method of the AndroidLuaCanvas object.
7. The Scheduler will then get a reference to the Surface of this node (which is the AndroidLuaSurface object created by the Lua player) and send it to the InterfaceUpdater module.
8. The Scheduler will set itself as listener to the adapter and call the *play* method of the Adapter, which will call the *play()* method of AndroidLuaPlayer.
9. Finally, in the *play* method of AndroidLuaPlayer:
 - The Lua Event module, is loaded into the Lua machine;
 - The reference to the previously created AndroidLuaCanvas is pushed to the LuaStack and set as global accessible variable named "mainCanvas".
 - The canvas.lua script, which defines the operations of the Lua canvas library in Lua code that access the AndroidLuaCanvas Java class, is loaded into the Lua machine. It verifies the existence of the mainCanvas object, and defines it as the root canvas.
 - It signalizes to the middleware that the presentation is beginning, by posting an event to all the Listeners.
 - And loads the script, which will be executed.

4.3.1 Validation Tests

In order to validate the implementation of the Lua canvas module, applications were developed to use the methods to draw to the screen. The NCL documents were simple enough just to provide a region for the Lua player and start the presentation. The results observed show the proper execution of the drawing calls, with overlaying objects according to the order of call. Figure 4.4 presents the graphical output of two simple applications, showing elements with different colors, and overlaying.

The associated NCL and NCLua scripts for the applications are similar to the example below.

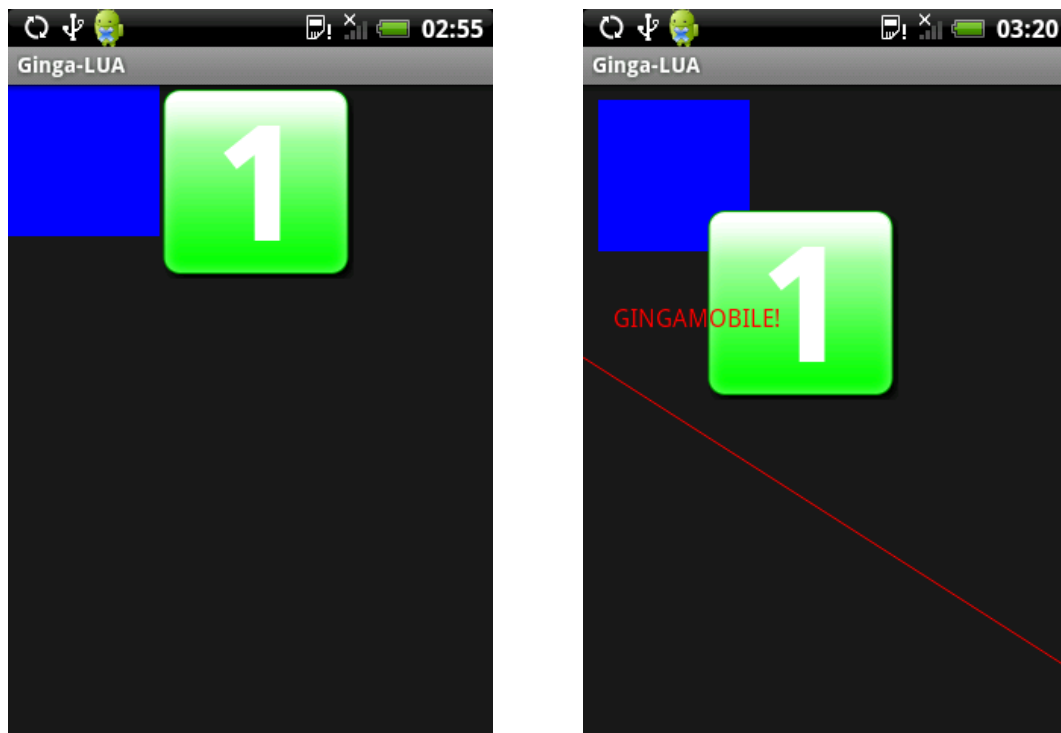


Figure 4.4: Graphical elements drawn from Lua

The NCL document defines a region, enclosing the whole display, with id "rg1", associated with the "ds1" descriptor. In the body of the document, the media component with id "lua1" associates the script file "1.lua" with the descriptor. The entry point of the body, defined by the port tag, is this media node.

<Aqui tinha um código>

This is the "1.lua" script. When it is executed, the Lua machine has already loaded the canvas library with a reference to the mainCanvas object, as was stated in the previous section.

The canvas object used here to perform the operations is the global canvas which has the reference to the main AndroidLuaCanvas object. The operations called, e.g. drawRect, are redirected to the AndroidLuaCanvas implementation. After all the drawing calls, a call to the canvas:flush() method is made, copying the buffer to the AndroidLuaSurface and displaying the contents on screen. After that, a timer is used to wait for 5 seconds and the post the presentation stop event, which will be sent to all the listeners of the AndroidLuaPlayerAdapter.

<Aqui tinha um código>

Chapter 5

Conclusion and Future Work

This work presented an improvement to the current implementation of Ginga-NCL for portable devices running the Android system. While still not fulfilling every aspect of SBTVD's standards, it is a clear large step into achieving that.

GingaMobile is currently the only open source implementation of Ginga-NCL for portable devices capable of running, given the constraints, a NCLua script from a NCL document. This result benefits both the academic research environment, that can study, improve and build upon this implementation, and to portable devices' application developers, that want to test their application even without a phone, as the middleware can be run inside the Android emulator.

One future work is, clearly, to achieve an implementation that is fully adherent to the standard. For this, the completion of the event module is necessary, as the implementation of the persistent and settings modules. It also still lacks a proper Tuner and Data Processing modules. There is now available in the market, Android phones capable of receiving the SBTVD signal (e.g. the Samsung Galaxy S Brazilian model). However, their pipeline has a proprietary API that is not exposed to the developers, and further work is needed to control the Tuner.

Another future work is to implement the Ginga-NCL middleware entirely using the Native Development Kit. This would allow sharing most of the code base of the current C++ implementation by PUC-RIO, which is updated constantly. This would most likely also lead to a better responsiveness of frame by frame drawing operations.

Bibliography

ABNT NBR 15601. *Digital terrestrial television - Transmission system*. 2008.

ABNT NBR 15602. *Digital terrestrial television - Video coding, audio coding and multiplexing*. 2008.

ABNT NBR 15603. *Digital terrestrial television - Multiplexing and service information (SI)*. 2008.

ABNT NBR 15604. *Digital terrestrial television - Receivers*. 2008.

ABNT NBR 15605. *Digital terrestrial television - Security issues*. 2008.

ABNT NBR 15606. *Digital terrestrial television - Data coding and transmission specification for digital broadcasting*. 2008.

ABNT NBR 15607. *Digital terrestrial television - Interactive channel*. 2008.

ABNT NBR 15608. *Digital terrestrial television - Operational guideline*. 2008.

AdMob. *Mobile Metrics Report*. May 2010. Available from Internet: <<http://metrics.admob.com/wp-content/uploads/2010/06/May-2010-AdMob-Mobile-Metrics-Highlights.pdf>>.

Android Developers. <http://developer.android.com/>. Last accessed on January 2010.

BRASIL. *Presidential Act Number 4.901*. 27 November 2003. Diário Oficial da República Federativa do Brasil. Available from Internet: <<http://www.jusbrasil.com.br/diarios/749358/dou-secao-1-27-11-2003-pg-7>>.

BRASIL. *Presidential Act Number 5.820*. 30 June 2006. Diário Oficial da República Federativa do Brasil. Available from Internet: <<http://www.jusbrasil.com.br/diarios/749358/dou-secao-1-27-11-2003-pg-7>>.

BRASIL. *Portaria Interministerial Number 237*. 29 December 2008. Diário Oficial da República Federativa do Brasil. Available from Internet: <<http://www.jusbrasil.com.br/diarios/6011627/dou-secao-1-30-12-2008-pg-93>>.

COMARELA, G. V. *Uma implementação do protocolo DSM-CC para plataforma Android*. 2009. Monograph presented for a B.Sc. in Computer Science.

DAHER, G. et al. Ginga-ncl for portable devices: An implementation for the android platform. In: *Proceedings of the XVI Brazilian Symposium on Multimedia and the Web*. [S.l.: s.n.], 2010. (WebMedia '10).

- Davic. *DAVIC 1.4 Part 2 - DAVIC Specification Reference Models and Scenarios*. 1998. Available from Internet: <<http://www.davic.org>>.
- ETSI TS 102 812. *Digital Video Broadcasting "Multimedia Home Platform (MHP) Specification 1.1.1"*. 2003.
- ETSI TS 102 819. *Digital Video Broadcasting "Globally Executable MHP version 1.0.2 (GEM 1.0.2)"*. 2005.
- FILHO, G. L. de S.; LEITE, L. E. C.; BATISTA, C. E. C. F. Ginga-j: The procedural middleware for the brazilian digital tv system. *Journal of the Brazilian Computer Society*, v. 12, n. 4, p. 47–56, march 2007.
- Gartner, Inc. *Gartner Says Worldwide Mobile Phone Sales Grew 35 Percent in Third Quarter 2010; Smartphone Sales Increased 96 Percent*. November 2010. Available from Internet: <<http://www.gartner.com/it/page.jsp?id=1466313>>.
- HAVi. *HAVi Level 2 Graphical User-Interface - Specification of the Home Audio/Video Interoperability (HAVi) Architecture*. 2001. Available from Internet: <<http://www.havi.org>>.
- ISO/IEC 13818-6. *International Organization for Standardization / International Eletrotecnical Committee, "Information Technology - Coding of Audio-Visual Objects - Part 6: Extensions for DSM-CC"*. 2000. 2000.
- ISO/IEC 14496-10. *International Organization for Standardization / International Eletrotecnical Committee, "Information Technology - Coding of Audio-Visual Objects - Part 10: Advanced Video"*. 2004.
- ISO/IEC 14496-3. *International Organization for Standardization / International Eletrotecnical Committee, "Information Technology - Coding of Audio-Visual Objects - Part 3: Audio"*. 2004. 2004.
- ITU-T Recommendation H.761. *Nested Context Language (NCL) and Ginga-NCL for IPTV Services*. April 2009.
- LUA.ORG. <http://www.lua.org/>. Las accessed on February 2011.
- MONTEZ CARLOS; BECKER, V. Tv digital interativa: Conceitos e tecnologias. In: *SBC (Org.). WebMedia e LA-Web 2004 - Joint Conference*. Ribeirão Preto, SP: [s.n.], 2004. p. 39–77.
- Open Handset Alliance. http://www.openhandsetalliance.com/press_110507.html. November 2007.
- SOARES, L. F. G.; BARBOSA, S. D. J. Tv digital interativa no brasil se faz com ginga: Fundamentos, padrões, autoria declarativa e usabilidade. In: CONGRESSO DA SOCIEDADE BRASILEIRA DE COMPUTAÇÃO. JORNADA DE ATUALIZAÇÃO EM INFORMÁTICA. Belém, 2008. p. 105–174.
- SOARES, L. F. G.; RODRIGUES, R. F.; MORENO, M. F. Ginga-ncl: the declarative environment of the brazilian digital tv system. *Journal of the Brazilian Computer Society*, v. 12, n. 4, March 2007.
- SOARES, V. M. C. e Marcio Ferreira Moreno e L. F. G. Ginga-ncl: Implementação de referência para dispositivos portáteis. In: *XIV Simpósio Brasileiro de Sistemas Multimídia e Web - WebMedia2008*. [S.l.: s.n.], 2008. p. 67–74.
- Sun Microsystems. *Oracle/Sun JavaTV: Java Technology in Digital TV*. 2008. Last accessed in January 2010. Available from Internet: <<http://www.oracle.com/technetwork/java/javame/tech/index-jsp-138011.html>>.

Annex

Annex A - Source Code of the Test Application

Aqui pode ter um anexo A.

Annex B - Source Code of the Lua Related Classes

Aqui pode ter um anexo B.