An Alternative Architecture for Ginga

Arturo Zambrano
Juan Antonio Zubimendi
Javier Búcar
LIFIA, Facultad de Informática
Universidad Nacional de La Plata
50 y 120
La Plata, Argentina
[arturo,azubimendi,jbucar]@lifia.info.unlp.edu.ar

ABSTRACT

The GingaNCL reference implementation is developed by PUC-Rio as a GPL project. This implementation runs on an x86 Linux virtual machine. The reference implementation is an excellent tool for communicating the intended behavior of Ginga but it is difficult to use it as the software running inside DTV dedicated platforms such as: set-top-boxes or integrated TV-sets.

There are two main issues that make it hard to port GingaNCL to dedicated platforms. First, GingaNCL has a monolithic structure that includes functionality usually provided by DTV dedicated platform such as: channel tuning, volume control and user menus. Second, the GPL license forces developers to make the changes available back to the community.

This paper presents a refactoring to GingaNCL that addresses the issues stated before by separating the tuning functionality, transport stream and data carousels processing on a different process and by defining an API between GingaNCL and the dedicated platform.

1. INTRODUCTION

Any successful software, eventually, ends up doing more things than it was originally intended for. As it evolves it may become a *big ball of mud* [5]. Avoiding such state implies constructing clear architectural boundaries. In order to recover architectural boundaries, large scale refactorings are needed. In this paper we will report on the result of such refactoring (not the refactoring process itself) that is an alternative architecture for Ginga.

Ginga is the Brazilian middleware for digital TV applications. It is composed by three main building blocks: Ginga CC, Ginga NCL [10] and Ginga J. The work we report in this paper is based on the reference implementation of Ginga NCL and Ginga CC as published by Pontificia Universidade of Rio de Janeiro (PUC-Rio, for short) on December 24th, 2009 and the results of changes made to that code base.

During our work on Ginga NCL we found some issues that prevented us from embedding it easily in commercial dedicated hardware platforms. We also found problems in the implementation of modules dealing with transport stream [6] (TS for short) and DSM-CC [7] data carousels processing. We reimplemented this functionality, and made some architectural changes. The code of GingaNCL is released under GPL license, therefore all modifications to it are released under the same license. The interested reader can find the source code of our derivative work at http://tvd.lifia.info.unlp.edu.ar.

This paper is organized as follows: Sect. 2 states the context and general objectives of the project where the refactoring has taken place. Sect. 3 presents an overview of the reference implementation of GingaNCL and some challenges derived from it. Sect. 4 presents the alternative architecture and discusses its consequences. We conclude the paper in Sect. 5 with the a summary of the contributions of this work.

2. CONTEXT

By the end of 2009, the Argentinian government entrusted LIFIA[1] the task of embedding Ginga NCL in a dedicated hardware platform. The government also required:

- A free, open source implementation of Ginga NCL.
- The provided solution must ease future embeddings for other platforms.

For the sake of clarity hereof we will make use of the following names:

Ginga NCL the specification of the presentation engine for NCL/Lua documents.

GingaNCL-PUC The reference GPL licensed implementation of Ginga NCL developed by PUC-Rio.

[1]LIFIA stands for Laboratorio de Investigación y Formación en Informática Avanzada. The lab is part of the College of Computer Science at the National University of La Plata. The web page of the lab is http://lifia.info.unlp.edu.ar

Ginga.ar It is our derivative of the reference implementation. Ginga.ar was forked from the original GingaNCL by January 2010.

3. OVERVIEW OF GINGANCL-PUC

The reference implementation of GingaNCL (GingaNCL-PUC) is developed by the Telemidia Lab at PUC-Rio. The source code of this implementation can be found at the Brazilian Portal of Public Software [3]. In this section we describe the modules composing GingaNCL-PUC implementation and their responsibilities. They are the starting point for our work.

3.1 Modules and Responsibilities

Below we list the modules present at the downloaded code base and their responsibilities:

telemidia-util This module provides an implementation of threads, utility functions for strings, colors, etc.

telemidia-links It is an adapted version of the Links [2] browser.

gingacc-cm The Component Manager is in charge of loading other modules in the system.

gingacc-system It hides platform details such as input methods, audio and video. Almost all other modules use gingacc-system

gingacc-ic Interactive Channel Manager provides access to the interactive channel (Internet connection), its implementation is based on Curl [1].

gingacc-um Update Manager: its implementation seems to be incomplete. As far as we understand it is a utility for handling Ginga upgrades, received through the interactive channel.

gingacc-player This module defines a player for each supported media object: LUA scripts, audio and video, XHTML, text, and images.

gingacc-tuner This module encloses tuning related functionality, such as: frequency change and transport streams reception (not processing)

gingacc-tsparser Transport Stream Parser is in charge of TS de-multiplexing, elementary stream filtering and section filtering.

gingacc-dataprocessing This module is responsible of processing EPG streams, data and object carousels (including events).

gingacc-contextmanager It handles user profiles and global configuration of the system.

ncl30 This module contains the entities that compose the NCL conceptual model [9].

ncl30-converter This module transforms NCL documents into the object model of NCL 3.0.

gingancl This is the NCL document formatter.

gingalssm Logical Subsystem Manager: this module is responsible for initializing the rendering device, creating the different players, and receiving and dispatching all the events (input and stream ones) to the players.

Figure 1 shows the main dependencies between the modules listed before. Only those modules relevant to this work have been included in the figure.

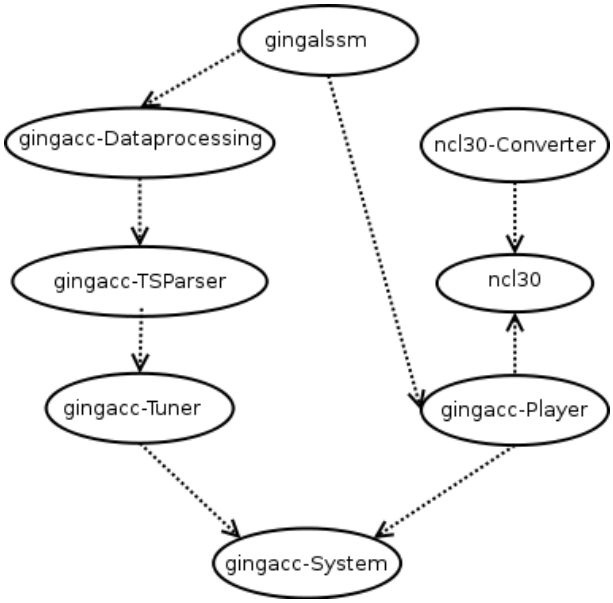

Figure 1: GingaNCL-PUC modules and relationships.

Note that gingalssm takes stream events from dataprocessing, we explain below how we cut and removed some functionality at this point.

3.2 Issues

After analyzing the code and making some tests we found the following issues that needed to be solved before continuing our work for embedding GingaNCL-PUC into the destination hardware.

3.2.1 Extrinsic Functionality

Looking at all these modules it can be seen that some of them belong to the core presentation engine and others provide services needed to feed this presentation engine – input for the presentation engine are the documents and the events. In particular some low level modules such as `gingacc-tuner`, `gingacc-dataprocessing` and `gingacc-tsparser` provide functionality that is unrelated to the presentation of NCL documents. For example, tuning functionality depends on the underlying hardware tuning services. Regardless which tuner is being used, for the NCL presentation engine what is paramount is the document and all the information related to it, this information includes all the files referenced by the document and the live editing events that affect the document's structure.

GingaNCL-PUC is well modularized at source code level. During the execution loading of modules is controlled by

the Component Manager. What we found here as an impediment is the fact that both document presentation and low level TS processing is done in the same process. We elaborate on this idea in Sect.4.1.

3.2.2 Scattered Implementation of the TS and DSM-CC Carousels Processing

Hiding design decisions is a key for good modularity [8]. With this idea in mind we review the implementation of TS and DSM-CC carousels processing. Transport stream processing is spread in `gingacc-tuner`, `gingacc-tsparser` and `gingacc-dataprocessing`. This fact makes it difficult to extend TS processing functionality.

Other problems we found in the implementation are:

- There was no support for carousel updates.
- Downloaded data was sometimes corrupt.
- It was not able to download carousels of more than 1MB. This prevented the execution of some applications, since carousels got never mounted.

After this analysis we decided to re-implement all this functionality from scratch.

4. ARCHITECTURE OF GINGA.AR

4.1 Motivation for Changes

In order to enhance maintainability of Ginga-NCL, it should be kept as simple and small as possible. In our opinion, the core of Ginga is responsible for document presentation but not for tuning or transport stream (data) processing. Tuning functionality and TS processing depends on the facilities that dedicated hardware and low level APIs provide, therefore they vary from one deployment to another, while the Ginga presentation engine should remain the same. The reference implementation of Ginga – the one provided by PUC – separates this functionality in the form of the three above mentioned modules (tuner, tsparser and data-processing) but once Ginga is built they belong to the same OS process. This means that a problem in one of this modules (that has nothing to do with NCL document presentation) affects the Ginga presentation engine. For example, a memory leak in the TS parsing can cause the OS to kill the process consuming memory, in this case it is same process that it is presenting interactive applications. Another example the case of a bug in the data carousel processing that causes a segmentation fault, which produces a crash that also kills the document presentation.

As we decided to re-implement all the TS and DSM-CC carousel processing functionality and we wanted to keep Ginga isolated and safe of our changes, we removed this functionality from the code base of Ginga. Thus the possibility of introducing bugs into GingaNCL during the reimplementation is minimized.

4.2 Alternative Architecture

Changes done to GingaNCL-PUC can be summarized into two main decisions, that we describe below.

4.2.1 Re-implement Transport Stream and DSM-CC Processing

Ginga.ar has a reimplementation of functionality regarding TS and DSM-CC carousels processing, including:

- Version checking for TS sections.
- Support for updates in data carousels.
- Support for large carousels, using modules up to 25400MB, tested size in the dedicated platform was 60MB.
- Other functionality not necessary for Ginga but a must for production set top boxes:
  - Support for SDTT tables.
  - Support for AIT.

4.2.2 Move Extrinsic Functionality to another Process

In order to have the presentation engine separated from the low level functionality mentioned before (Sect. 4.2.1), we removed it. Now low level functionality is carried out in another process, called Zapper[2]. The Zapper provides basic TV functionality such as: frequency and program changing, volume adjustment, scanning of digital TV channels, etc, and now also the TS data processing.

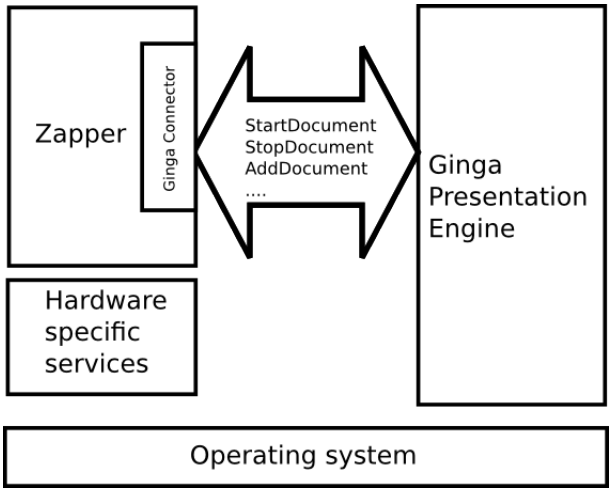

Figure 2: Architecture of Ginga.ar (Ginga presentation engine and Zapper processes)

Changes made to the code base lead us to an alternative architecture, which is depicted in Fig. 2. A new Zapper process runs on top of the operating system and libraries for accessing hardware specific services (such as hardware accelerated rendering, tuning, etc). The wide arrow is a bi-directional communication channel between the two processes. At implementation level it is a Unix domain socket. Instead of the interaction between the `gingalssm` and `dataprocessing` explained in Sect. 3.1 we added a protocol between the Zapper process and Ginga presentation engine. This communication

[2]Normally zappers does not perform data processing, we are using the word "zapper" extending the typical meaning.

protocol allows the Zapper to command the Ginga presentation engine. Currently, we are using this architecture to deploy GingaNCL into a production set top box. Note that there is a module called Ginga Connector, which is embedded into the Zapper. This module communicates Zapper with Ginga presentation engine.

Note that Ginga presentation engine is GPL as it is a derivative of GingaNCL-PUC, in contrast Ginga-Connector is licensed under LGPL. We explain the reason and implications of this in Sect. 4.3.3

As interactive applications are transmitted using data carousels, they are received in the set top box by the Zapper process. It mounts the data carousel. When an event is received, indicating that application needs to be started, Zapper – through Ginga-Connector– sends a command to Ginga presentation engine to add the document and start playing it. If further editing events are received, new commands are sent to the Ginga presentation engine process.

Zapper is launched upon system boot. Once an interactive application is received the Ginga-Connector launches Ginga presentation engine. After the engine is up, it is instructed to run the application. Connector can send after that a kill or destroy application command (or others commands) according to the AIT events received. Another possibility is to launch both processes after boot, and instruct Ginga presentation engine to start or stop documents as needed. In the current deployment we choose to launch Ginga only if an interactive application has been received.

Ginga presentation engine is able to invoke functionality on the Zapper side (through Ginga-Connector), for example, to reserve control remote key events that must not be interpreted by the zapper during the execution of a given interactive application. Ginga presentation engine also relies on the zapper for some media presentation functionality, for example main video re-size or positioning. Summarizing, the communication protocol between both process allow bidirectional communication.

4.3 Consequences

Here we describe some benefits and drawbacks obtained from the current architecture.

4.3.1 *Quick Interactive Application Start*

The presented architecture allows to have Ginga presentation engine process running, even when no interactive application is present. This contrasts with a schema where the presentation engine is started when an application must be executed. Ginga.ar allows to immediately execute the applications, since when an application is received and its start is signaled in the transport stream, the Ginga presentation engine is up and ready to execute it.

4.3.2 *Easier Testing*

In Ginga.ar, the presentation engine is commanded from Ginga-Connector. It is easy to build a testing oriented *connector* that allows to test the presentation engine. Many stress and stability tests in Ginga.ar were conducted using a scriptable *connector*.

4.3.3 *License Schema Suitable for both Ginga Evolution and Embedding*

Ginga.ar is GPL licensed, as it is derivative work of GingaNCL-PUC. We are for GPL software and one of our objectives is to help the evolution of GPL Ginga. Nevertheless, we are aware that, generally, the APIs provided by hardware manufacturers are proprietary. To overcome this gap, the architecture provides the following solution: Ginga Connector is LGPL and GPL licensed. It allows to keep the Zapper's code closed but at the same time it allows to make use of a GPL licensed Ginga. At implementation level this done communicating both processes through a Unix domain socket. Both Ginga presentation engine and Zapper are linked against the Ginga Connector –it is a shared library–, which define the structures for messages. Zapper is never linked against Ginga.

Plans for the future includes to release a GPL Zapper, meanwhile, the LGPL Ginga Connector will help to keep Ginga evolving to a mature product.

4.3.4 *Obliviousness of the Events Transmission System*

Events affecting an interactive application can be encoded as AIT [4] control commands or editing commands (stream events). Having an homogeneous protocol for interacting with the Ginga presentation engine abstracts the engine from the underlying format or transmission method of events.

4.3.5 *Resources Consumption*

Having two processes cause more resource consumption. Also, the communication protocol between the processes add overhead to the computation (compared to a single process architecture).

4.3.6 *Process Status Synchronization*

Having the two processes adds complexity. The Zapper or Ginga-Connector must be aware of the state of the Ginga presentation engine. For example: if the engine freezes, Ginga-Connector is responsible for killing the engine and starting a new one.

5. CONCLUSIONS AND FUTURE WORK

In this work we have analyzed the modules composing the reference implementation of GingaNCL. We also presented the architecture of Ginga.ar, a derivative of GingaNCL-PUC. The main benefits of the new architecture, whose purpose is to speed up the process of embedding Ginga in different hardware platforms have been discussed. The distinctive characteristic of the presented derivative architecture is the separation of functionality into two different processes (one for transport stream and DSM-CC processing and other the NCL presentation presentation engine).

Future work on Ginga.ar include the replacement of Links web browser and large scale refactoring of its threading model.

6. ACKNOWLEDGMENTS

First of all, the authors want to thank the researchers of Telemidia Lab at PUC-Rio, most notably Prof. Luiz Fernando Gomes Soares and Marcelo Moreno. They develop

Ginga NCL reference implementation, make it free and open source, and always have excellent predisposition for collaborating with us.

The authors thank also Federico Balaguer, Martin Olivera and Ignacio Jaureguiberry for their useful comments on drafts of this paper. Finally, the authors want to thank Richard Matthew Stallman for his cooperation in the definition of a valid licensing schema for Ginga.ar.

7. REFERENCES

[1] curl groks urls. http://curl.haxx.se/.

[2] Links. the www text browser. http://links.sourceforge.net/.

[3] Portal do software público brasileiro. http://www.softwarepublico.gov.br/.

[4] Associação Brasileira de Normas Técnicas. *Digital terrestrial television — Data coding and transmission specification for digital broadcasting. Part 3: Data transmission specification*, August 2008.

[5] B. Foote and J. Yoder. Big Ball of Mud. *Pattern Languages of Program Design 4*, pages 277–288, 2000.

[6] International Organization for Standardization. *Generic coding of moving pictures and associated audio information Part 1:Systems*, 2007.

[7] International Organization for Standardization. *Generic coding of moving pictures and associated audio information. Part 6: Extensions for DSM-CC*, 2007.

[8] D. L. Parnas. On the criteria to be used in decomposing systems into modules. *Comm. ACM*, 15(12):1053–1058, Dec. 1972.

[9] L. F. G. Soares and R. F. Rodrigues. Nested context model 3.0. Technical report, PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO DE JANEIRO, Departamento de Informática, RUA MARQUES DE SÃO VICENTE, 225 - CEP 22453-900 RIO DE JANEIRO - BRASIL.

[10] L. F. G. Soares, R. F. Rodrigues, and M. F. Moreno. Ginga-ncl: the declarative environment of the brazilian digital tv system. *Journal of the Brazilian Computer Society*, page 37–46, 2007.