HAL open science

Ginga-J - An Open Java-Based Application Environment for Interactive Digital Television Services

Raoni Kulesza, Jefferson Lima, Álan L. Guedes, Lucenildo Junior, Silvio Meira, Guido Filho

To cite this version:

Raoni Kulesza, Jefferson Lima, Álan L. Guedes, Lucenildo Junior, Silvio Meira, et al.. Ginga-J - An Open Java-Based Application Environment for Interactive Digital Television Services. 9th Open Source Software (OSS), Oct 2011, Salvador, Brazil. pp.34-49, 10.1007/978-3-642-24418-6_3 . hal-01570756

HAL Id: hal-01570756

https://inria.hal.science/hal-01570756v1

Submitted on 31 Jul 2017

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Distributed under a Creative Commons Attribution 4.0 International License

Ginga-J - An Open Java-based Application Environment for Interactive Digital Television Services

Raoni Kulesza[1,2], Jefferson F. A. Lima, Álan L. Guedes, Lucenildo L. A. Junior[1], Silvio R. L. Meira[2], Guido L. S. Filho[1]

[1]Laboratory of Digital Video Application (LAVID)
Federal University of Paraiba (UFPB)
Cidade Universitaria - 58059-900,
João Pessoa/PB, Brazil
{raoni, jefferson, alan, lucenildo, guido}@lavid.ufpb.br

[2]Informatic Center (CIn)
Federal University of Pernambuco (UFPB)
Cidade Universitaria - 50740-560,
Recife/PE, Brazil
srlm@cin.ufpe.br

Abstract. This paper aims to present a Ginga-J's reference implementation. Although based on a particular platform, the implementation not only works as a proof of concept, but also raised several issues and difficulties on the software architecture project that should be taken into account to ease extensibility and porting to other platforms. Ginga is the standard middleware for the Brazilian DTV System. Its imperative environment (Ginga-J) is based on new JavaDTV specification and mandatory for fixed terrestrial receptors

1. Introduction

With arise of the Digital TV a new set of functionalities were incorporated to the shows offered by stations as well as to the DTV's receptors. Therefore, the TV environment has become more interactive, as TV systems (or middleware) now offer an environment for the execution of interactive applications. These applications can be transmitted and executed along with multimedia content such as audio, video, image, text, etc, thus enabling the increase of interactivity between viewers and the television through applications such as games, polls, etc. [1]. All those services or applications could not be possible without the support of an intermediary software layer, called middleware, installed on each access terminal.

The execution of a same application in different devices with distinct processing capacities and hardware architectures is achieved through the specification of well-defined software architecture. The main role of a Digital TV middleware is to act as an intermediary software layer between the operating system and the interactive applications, abstracting specific characteristics of the platform and providing a series of specific services to the above layers. Thus, it is possible the developing of portable

applications to many distinct receptors.

The main DTV middleware open specifications offer support to the execution of interactive applications in two environments: a declarative and an imperative [2]. On the Brazilian Digital TV System, the declarative environment is represented by the Ginga-NCL [3][4], which supports applications based on NCL language (Nested Context Language) and the imperative environment is represented by Ginga-J [5], which provides support to the execution of applications written in Java language.

This paper main goal is to present the first implementation of Ginga-J, highlighting its singularities when compared to other middleware's implementations. On [5] is described all the information about the Ginga-J's functionalities specification. This article describes the reference implementation on the GingaCDN's[1] project context in order to serve as basis to future Ginga-J's implementations from different manufacturers and its platforms. Another point discussed in this article is the evolution and validation of Ginga-J architecture's reference implementation, since it is based on preceding work on middleware developing, realized by the same research group from LAVID at UFPB [6] . The main results from the work shown here were: (i) definition of a flexible architecture that allows reuse and software extensibility and; (ii) developing of a reference implementation in conformity with the new JavaDTV API recently adopted by the Ginga middleware.

This article is organized as follows: section 2 describes the Ginga middleware. The section 3 presents Ginga-J's specification history. The section 4 talks about the implementation architecture proposed for the Ginga-J. The Section 5 details the developed implementation. The section 6 discusses the main existing DTV's middleware projects for fixed terminals, performing a brief comparison between these middlewares and the Ginga-J. And, lastly, section 7 presents the final considerations, future and current works.

2. Ginga Middleware

Ginga is the SBTVD's middleware specification, it resulted from the fusion of FlexTV [6] and MAESTRO[7] middlewares, developed through a consortium led, respectively, by UFPB and PUC-Rio on the SBTVD [8] project.

The FlexTV, procedural middleware proposed by SBTVD's project, included an API set compatible with other standards along with novel functionalities such as the possibility of communication with multiple devices, allowing different viewers to interact with the same interactive application using remote devices. The MAESTRO was the declarative middleware proposal of SBTVD's project. Focusing on space-time synchronization between multimedia objects using the NCL (Nested Context Language) declarative language combined with the functionalities of the scripting language Lua.

Ginga combined these two solutions, now called Ginga-J and Ginga-NCL, considering the ITU's international recommendations [11]. Thus the Ginga middleware is divided in two main interconnected subsystems (Figure 1), also known

[1] GingaCDN Project. available on http://www.openginga.net

as Execution Machine (Ginga-J) and Presentation Machine (Ginga-NCL). The imperative content execution is possible through the Java Virtual Machine (JVM). Depending on the application requirements, one programming paradigm can be more appropriate than other.

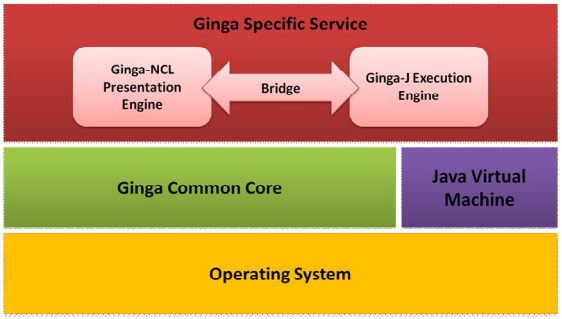

Figure 1 - Overview of Ginga middleware

Another important aspect is that the two environments, for the execution of interactive applications, are not necessarily independent, since that ITU's recommendation includes a "bridge", which provides mechanisms for the communication between them. This bridge API allows imperative applications to use available services on declarative applications, and vice versa. Therefore the execution of hybrid applications one level above the layer of execution and presentation environments is made possible, allowing to combine the NCL language facilities of multimedia elements presentation and synchronization with the power of the object oriented Java language.

Ginga Common Core is the Ginga subsystem responsible for providing specific functionalities of Digital TV common to the imperative and declarative environments, abstracting the specific characteristics of platform and hardware for the above layers. Some of its main functions we can mention are: media exhibition and control, system resources control, return channel management, storage devices, access to service information, channel tuning, among others.

3. Ginga-J specification

The Ginga-J (Figure 2) is composed by a set of APIs, defined to provide all the necessary functionalities for the developing of DTV applications, from the multimedia data manipulation, to access protocols. Its specification is formed by an adaptation of the information access API of the Japanese standard service (ISDB ARIB B.23), the Java DTV [12] specification (which includes the JavaTV API), besides an additional set of extensions or innovation APIs.

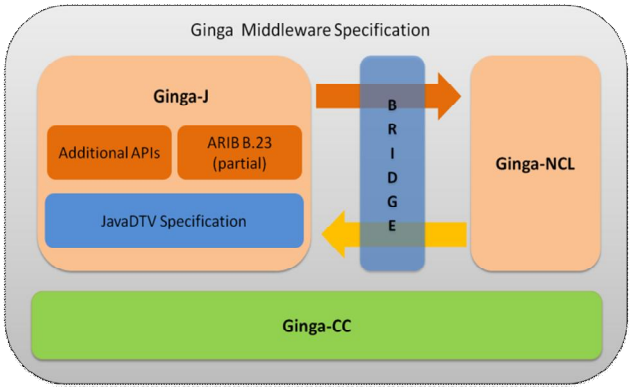

Figure 2 - Ginga-J overview

The additional APIs include a set of available classes for the bridge between applications written in NCL and Java language, additional functionalities for tuning channels, sending asynchronous messages through the interactivity channel and integration of external devices, enabling the support to multimedia resources and simultaneous interaction of multiple users on DTV [13] applications.

The Java DTV [12] specification is an open and interoperable platform that allows the implementation of interactive services in Java language, which has been recently adopted to the Ginga-J's APIs set. Functionally, the JavaDTV replaces the APIs collection that was previously used and defined by the GEM standard (Globally Executable MHP), such as DAVIC (Digital Audio Video Council) and HAVi (Home Audio Video Interoperability). The goal was to provide royalties free solution for device manufacturers and application developers, allowing the production of TV sets and/or set-top-boxes at an affordable cost.

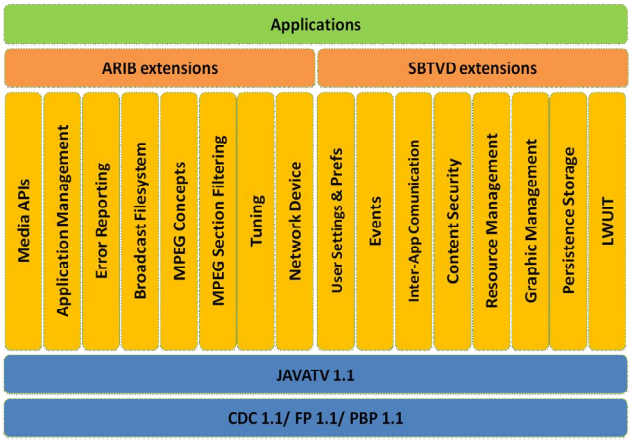

Figure 3 - Ginga-J's APIs set

The current specification is composed by the Java DTV and JavaTV APIs, added to the Java execution environment (Java Runtime) for embedded systems (JavaMe),

including the CDC platform (Connected Device Configuration), and the profile APIs: FP (Foundation Profile) e PBP (Personal Basis Profile) (Figure 3). Among the main differences of Java DTV related to the application development, we can quote the LWUIT API (LightWeight User Interface Toolkit), responsible for defining graphic elements, graphic extensions for DTV, layout managers and user events.

4. Reference Implementation Architecture

The specification of Ginga-J's reference implementation architecture was based on the FlexTV [6] architecture, which considered the J.200 ITU [11] architecture. However, besides following a different set of APIs definitions (based on JavaDTV, not GEM), other features were added to provide better reuse and software quality. The Figure 4 illustrates the modularized conceptual architecture: (i) operating system, (ii) common core layer and; (iii) Ginga-J's execution machine. Following are described the three stages which were adopted to define the architecture solution.

The first step for the architecture's definition was to choose a suitable execution platform for the characteristics and differential limitations of a Digital TV fixed terminal. With that in mind, we chose the Linux operating system for personal computers (x86) and the PhoneMe Java [15] virtual machine, which is an official implementation of JavaME/CDC's environment. The main reason of this choice was the Linux and PhoneME availability as open platforms, and also the offer of many development tools without additional cost. Besides, Linux supports heterogeneous systems [16]. The aim was to allow the implementation's development on an environment closer to an access terminal, but that could also be available to as many developers as possible. In this case, a personal computer, without the need to buy any specific hardware.

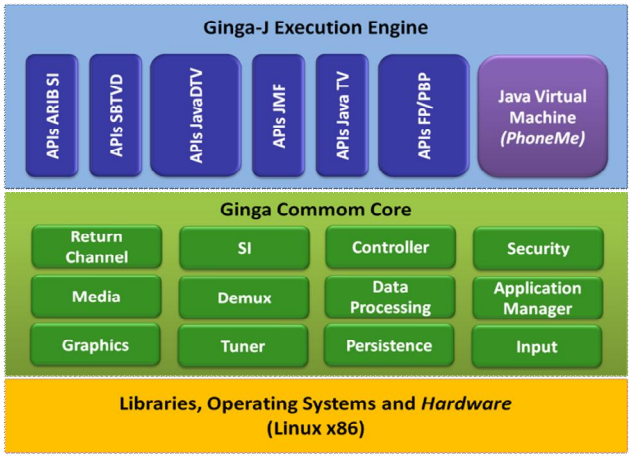

Figure 4 - Conceptual Architecture

The second stage was to develop and refine FlexTV's common core architecture. Nearly no change was performed in the conceptual definition of these subsystem's modules, there was merely a refactoring in order to attain better functionalities

cohesion. The main change was to specify the common core using a component-based approach, adopting a component model and an execution environment: FlexCM[17].

The goal was to emphasize the software modeling by decomposing the system in functional components with well-defined interaction interfaces. In this context, a component model defines the instantiation scheme, composition, life cycle of the system components and an environment of software execution responsible for managing the components ensuring the specifications defined by the respective components' model.

The FlexCM model follows a declarative approach, in which the components define its dependencies explicitly (required interfaces) and the execution environment loads and provides the dependencies through a dependencies' injection standard. The FlexCM model allows its components to know only the interfaces; the implementations are treated through the execution environment. Besides the required interfaces, the components can also declare configuration parameters which values are also injected through the execution environment allowing the developer to easily configure the component in the final product where it will be installed. The FlexCM's execution environment is capable of loading the entire system from an architectural description file in which the connections and configurations are specified.

The adoption of the FlexCM's components model offered a series of specific advantages for Ginga-J's implementation besides the commonly known advantages for a component based development, like modularity, maintainability and reuse we can quote: (1) knowing the architecture in model level; (2) facilities on the configuration of individual components and; (3) on the system configuration as a whole. Lastly, these characteristics bring the possibility of managing different architectures also easing the execution of unit tests and integration of different portions of the architecture. A test process proposal for the Ginga-J based on FlexCM can be found at [18].

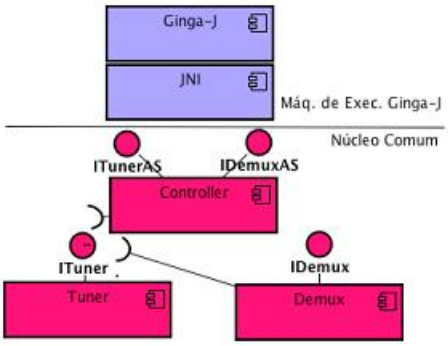

Figure 5 - Ginga-J Execution Layer and Common Core integration

The third and last stage was to define an integration model of the Common Core layer with the Ginga Execution Machine. As mentioned, the Common Core is responsible for offering services for the Ginga-J execution machine. Consequently, it contains

native code (in C or C++ languages) and it depends on the platform's execution libraries. It was then important to define a communication model in order to reduce the coupling and the dependency between these two subsystems. The adopted solution was based on the *Proxy*, *Facade* and *Adapter* [19] design patterns. The idea here was to centralize all Java execution machine use on a Controller module, which exposes the services for the applications (Application Services). Figure 5 illustrates the module Controller with two AS interfaces: `ITunerAS` and `IDemuxAS`. These services are offered for the Ginga-J's applications through JNI (Java Native Interface) callings implemented internally through Java's packages. The Controller calls by delegation the component that implements the required functionality. For example, `ITuner` and `IDemux` calls (shown on Figure 5). If a Common Core component needs another Common Core component functionality, it can call it directly. The main advantage of this approach is to isolate the layer(s) above the Common Core, in such way to prevent platform dependencies, as well to decrease the coupling between Java API's specifications and the implementation in C/C++ code. For example, the port of a Ginga-NCL's presentation machine or a Java's execution machine from another Digital TV (GEM) system to the Common Core used in this work would be facilitated.

5. Implementation

In this section the Ginga-J's implementation is described focusing on its Common Core components. Figure 6 displays this subsystem overview, which contains the following components:

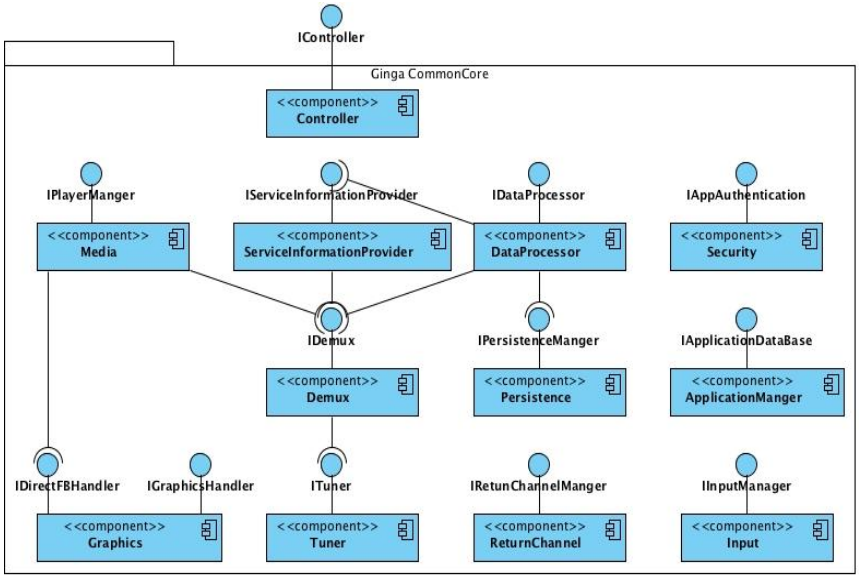

Figure 6 - GingaJ's Common Core implementation (in this case, Controller is not a component, just a facade)

(1) **Tuner** - tunes and controls the access to the multiple network transport streams; (2) **SI** – obtains service information from the transport stream, in other words, which elementary streams (semantics) of audio, video and data has been transmitted, besides information as parental rating, synopses and time scheduling; (3) **Demux** – provides specific filters to select streams; (4) **Media** - Provide access to media decoders (hardware and software) to manage and display the presentation of video and audio elementary streams; (5) **Data Processor** – processes and separates transmitted data (e.g. applications) in multiplexed MPEG-2 transport streams; (6) **Graphics** – provides graphical user interface handling; (7) **Input Manager** – handles user key events through the remote control, STB's panel keys, keyboard, or another input device; (8) **Return Channel** – provides interfaces for the return channel's usage, for example, through dial-up, ADSL, Ethernet, WiMax or 3G; (9) **Application Manager**: loads, instantiates, configures and runs applications; (10) **Persistence** – manages non-volatile storage resources; (11) **Security** – verifies an interactive application's authenticity and permission; (12) **Middleware Manager** – responsible for the middleware's functional management.

As previously mentioned, for Ginga-J's execution machine reference implementation the RC2 version of Linux's PhoneMe Advanced was used [15]. A native port of the Java AWT graphic API for the DirectFB[2] was performed in the virtual machine. The generated code was based on the PhoneME built-in native implementation in Qt. Then, it was possible to implement the JavaDTV APIs, using the Java's base classes, which are present in PhoneME, for example, the graphical interface API and user events handling API. These functionalities were encapsulated in Graphics and Input components, respectively. For functionalities not present on the Java environment it was necessary integration with the Common Core. To allow Java applications management, it was also necessary to integrate the JVM with the Common Core through the `Controller` and `ApplicationManager` components. The `Controller` component implemented a new proxy element, which enabled the execution of *Xlets* through the `ansiJavaMain()` `fuction` (available on JVM's code). This function starts the JVM and runs a Java class that initiates all graphical layers available (as specified on SBTVD's standard), and also loads the interactive applications' data (*Xlets*) which are started as separated *Threads*, since Ginga allows the execution of more than one application at the receptor.

The `Tuner` implements required services of the `com.sun.dtv.tuner` package, using a scanning process for identifying non-blocking channels based on events and on the Observer pattern[19].

The `Demux` component contains functionalities from the `com.sun.dtv.filtering` package, allowing the selection of different types of elementary streams. Internally it uses a "circular queue buffer" with different start pointers, one to feed each user, trying to avoid that users lose their contents consumed by others.

---

[2] DirectFB is an open source project which provides graphic acceleration, input events treatment, graphic layers management and reproduction of several medias through multimedia providers: Available on: http:// www.directfb.org

The SI component obeys the APIs' requirements which deals with service information (ARIB, JavaTV and JavaDTV, besides allowing the component user to obtain final information about the stream, without the need of another processing, since it implements a *cache* mechanism. All the abstractions for Service Information provided by SBTVD's standards [20] (Table, Descriptors and Events) can be generated from an object factory, which uses the *Factory Method* pattern [19]. This component also warns the DataProcessing to perform the signalization, execution of applications and data carousel.

The Media component is responsible for the middleware's processing of continuous media (audio and video) received from Demux using the vlc infrastructure to present the media over a DirectFB surface. Acquiring validation of the implementation with a performance analysis [27]. This component was designed considering the requirements of the JMF API, since it provides basic reproduction functionalities for the Java API through the JNI calls.

The ApplicationManager offer interface abstraction for applications in your database, this abstraction is called ApplicationProxy, witch offer the control of the applications lifecycle (start, stop, pause, resume and destroy). A example, JavaProxy is a child class of ApplicationProxy, that has the capabilities to call the ansiJavaMain() function to start the a xlet. As the same, also exist the NCLProxy that has de capabilities to start a NCL Presentation Engine [10] to start a NCL document.

Considering the execution of the applications in deferents process, the ApplicationProxy must use IPC(Inter Process Communications) strategy to send commands to your engine execution, example send events received through the InputManager or a control command.

Beside the lifecycle of the applications, the each Proxy`s interface contains functionalities to offer communication inter applications. In Java Engine, this happens through javax.microedition.xlet.ixc for interaction with another xlet, and br.org.sbtvd.bridge for control NCL documents.

The Persistence and Security components work together to strictly follows the JavaDTV[12] model to pack, authenticate and authorize the applications and file storage. That consist in persist the jar file of the application and study the application access permissions in platform.

The Persistence component has important interaction with the ApplicationManger component, given that the last send destroy events when a execution of a application is finished, this provides the trigger to Persistence deallocate the finished application resources.

The Return Channel component implements the TCP/IP communication for different network technologies, offering abstraction about the orientation to connection in two types ConnectionReturnChannel(dial-up, ADSL, 3G) and ReturnChannel(Ethernet, WiMax). The Return Channel and the Persistence component acquired validation of your implementation by used in LARISSA project[26]. The Figure 7 below illustrates 4 (four) use scenarios of Ginga-J's

implementation.

The Figure 7(a) displays an Java (*Xlet*) application using the access APIs for Service Information (JavaTV SI and ARIB SI) and Ginga-J's graphic elements (LWUIT) APIs. The Figure 7(b) shows an application displaying 3 video streams (2 locals and 1 live) as a validation scenario for the implementation of the media's execution API (JMF). Now the Figure 7(c) and the Figure 7(d) illustrate the possibility to execute a Java application from a local file (for example, USB device) or from a transport stream, respectively. So, as on a TV set, many middleware configurations can be modified through an OSD resident application (*On Screen Device*). The two last examples supported the APIs' validation for the lifecycle control of the application (JavaTV), data carousel, persistence and security (JavaDTV). The tests were conducted using a personal computer with the following specifications: Core 2 Duo 2.16GHz processor; 1GB RAM; operating system Ubuntu 9.10 Kernel 2.6.31-14, and; a 100 GB hard drive.

Figure 7 - Ginga-J's use Scenarios

6. Related Works

The main existing middleware's implementations on the Digital TV context might be divided in two categories: (1) declarative environments (2) imperative environment. The first group is represented by: (i) LASeR(*Lightweight Application Scene Representation*) [21]; (ii) BML (*Broadcast Markup Language*)[22]; (iii) GingaNCL for portable devices[23] and; (iv) Ginga-NCL for fixed devices[24]. However, for the second, we can quote: i) FlexTV[6] and (ii) OCAP-RI (*OpenCable Application Platform – Reference Implementation)*[25].

On [23] it was presented a comparative analysis between the solutions LASeR, BML and Ginga-NCL for portable devices. The main difference of these solutions regarding the implementation proposed here (Ginga-J) is at the architecture project. None of these solutions uses a component-oriented approach, not defining a model and execution environment for the system modules. Besides, we can observe that these environments seek to implement the following functionalities: medias' synchronization, adaptability, support of multiple devices, supports on air edition, and

also supports reuse. The Ginga-NCL for portable devices is the only solution that supports multiple devices and meets reuse support standards. The solution proposed here also attends all the requirements presented by the declarative environments, but uses an imperative approach, through the object-oriented language Java. The use of this kind of language is much harder and susceptible to errors and also requires a bigger *footprint* from the application. Nonetheless, it carries a power of expression larger than that offered by declarative languages. The goal is to offer more advanced applications that need to use, for example, access and security mechanisms, finer control to information and audio-visual content.

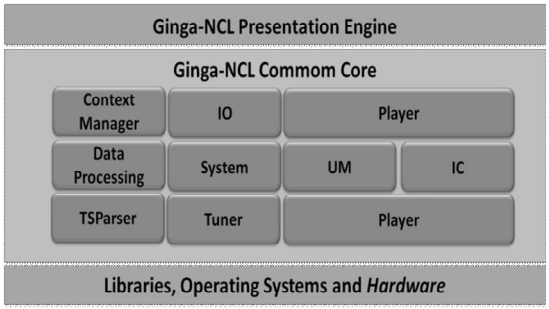

Figure 8 - Overview of Ginga-NCL's fixed devices

Figure 8 displays an overview of the implementation for Ginga-NCL's fixed devices [24]. Ginga-NCL's Presentation Machine is also a logic subsystem capable of starting and controlling NCL applications. Ginga-NCL's Common Core is responsible for offering the previously mentioned services for Ginga-NCL's Presentation Machine. This solution, although attending a different set of applications, displays further similarity on the definition of Ginga-J's Common Core functionalities. One of the differences is on the absence of security functionalities and a lower set of informations about the offered service. The `Tuner`, `DataProcessing`, `ContextManager`, `InputOutput (IO)` e `InteractiveChannel (IC)` components of Ginga-NCL, are equivalent, respectively, to `Tuner`, `DataProcessing`, `ApplicationManager`, Input and `ReturnChannel` of Ginga-J. Media and Player of Ginga-J represent functionalities of Ginga-NCL's Player module. `Demux` and `SI` Ginga-J modules represent the Ginga-NCL's `TSParser`. The module *`System`* of Ginga-NCL is implemented internally on GingaJ. The main reason for representing Ginga-J's functionalities with more modules is to allow better cohesion and, consequently, larger extension flexibility and code maintenance. Another important difference concerns the implementation on the modules' management mechanism. On Ginga-NCL this is implemented by `ContextManager` and `UpdateManager` (*UM*) and on Ginga-J a model and execution environment of software components (FlexCM) are defined.

Table 1 - Comparison between Ginga-J and Ginga-NCL for fixed devices

	Ginga-J	Ginga-NCL for fixed devices
Creation model	Dependency injection	Factory
Architecture's knowledge	On model's level	On source's level
Life cycle	Creation, initialization, pause and destruction	Creation and destruction
Components configuration	Standard model	Absent

Table 1 displays a comparison of the solutions. On the criteria for evaluation, we observed that the Ginga-NCL model uses an approach of object factory, imposing that the architecture knowledge is spread through the system's source code. This characteristic limits the flexibility in which the architecture may be instantiated. Besides, the lack of standardization in order to configure the components prevents an effective management of the system modules. So it is believed that the model used on Ginga-J's implementation best meets the requirements for modularity, maintainability and reuse of the project and implementation of the Common Core's code.

The FlexTV implementation was realized by Ginga-J's same group and the current proposal is an evolution of the same in two points: (1) functionalities (new set of APIs Java based on JavaDTV) and (2) architecture (adoption of a model and environment of components execution).

OCAP-RI Moreno, F. M. A Declarative Middleware for Digital TV Systems. (Master Thesis); PUC-Rio, DI, 2006

[25] is a proposal of imperative middleware implementation based on the American standard of Digital Cable TV. One of the differences is on the set of offered functionalities, fewer than Ginga-J, since OCAP's Java APIs do not support multiple devices nor users, management of the multimedia streams and asynchronous messages. Another important point is related to the architecture project (Figure 9) which is divided into: (I) OCAP Java – set of Java APIs available for applications and defined by the TVD American standard; (ii) JVM and OCAP Native – Java's virtual machine and set of specific native code to implement OCAP Java's functionalities; (iii) MPE *(Multimedia Platform Extensions*) – layer that abstracts the execution platform for the JVM and the OCAP Native code; (iv) MPEOS *(Multimedia Platform Extensions Operating System)* -implements platform dependent code offering services for the MPE, which means, MPEOS is the code that needs to be ported for each platform and; (v) *RI Platform* – represents the operating system and the hardware that runs the middleware. The MPE and MPEOS layers from OCAP-RI are equivalent to the set of components of the Ginga-J's Common Core, where MPE is represented by the interfaces of *Controller* and MPEOS by internal implementations of each component. As already quoted, such characteristic facilitates the port of the Java execution machine for different platforms. However, MPE and MPEOS are implemented using C language and do not use any model and components execution environment. Therefore the OCAP-RI architecture does not offer any modulate division of functionalities, making reuse and code flexibility more difficult.

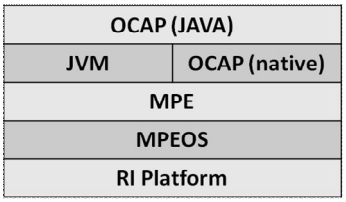

Figure 9 - OCAP-RI's Architecture

Based on the points discussed in this section the main differences between Ginga-J's implementation and other proposals can be understood. The first is related to the programming model and the set of different functionalities offered by an imperative environment in relation to declarative environments or imperative environment based on GEM. The second refers to the architecture project, which tried to attend reuse, maintenance and code flexibility the best way possible. Such aspects are important for the implementation of reference, since itself offers a starting point and can be adapted to many platforms by manufacturers and other developers.

7. Development Process

The GingaCDN (Ginga Code Development Network) was idealized as a group of developers and contributors (scattered across the globe) of Brazil's Digital TV middleware the Ginga. Among the various projects being carried out by this network is Ginga-J reference implementation. Nowadays, the number of registered developers reached 570 from 15 different countries on GingaCDN community site.

In order to become a distributed software development team and gain from its benefits like reducing costs and time spent, while improving the software's quality, it was necessary to thwart it's known drawbacks such as inefficient communication, loss in coordination and providing a full vision of the project. The solution came through a well-defined development process with roles and a proper tool (based on Redmine[3] tool) to support it all.

The collaborative development process was designed to standardize agile and objective practices as to attend the deployment of distributed component for digital TV's middleware. Thus, we defined roles for the collaborative team members in order for users to know their responsibilities and have the freedom to attend their assigned tasks independent and simultaneously. Such roles are distributed in the five phases defined in the collaborative development process. Each phase of the process is mainly conducted by a specific role (except for the Review phase), they are: 1) Conception, where the Manager creates a new subproject; 2) Elaboration, when the Leader specifies tasks to accomplish the subproject; 3) Construction, is carried out by the Developers undertaking the tasks; 4) Review, the Reviewers review the component and Integrators check if they integrate with the whole project; 5) Transition, the Manager once again comes along to check if the component is in accordance to what was out lined initially.

[3] Redmine Project: Available at: http://www.redmine.org

8. Final Remarks

This work describes the Ginga-J's reference implementation, SBTVD's imperative middleware. The development was based on JavaDTV's specification, an architecture based on software components. As a form of proposal validation, the architecture was instantiated for the Linux environment on a personal computer. The main Java packages of Ginga-J's standard were implemented through the integration of basic Java's environment (*PhoneME*) functionalities as well as implementations of specific functionalities for Digital TV (Ginga-J Common Core).

The project and implementation of a development based architecture using components brought a series of benefits, such as: (i) knowing the architecture at the model level; (ii) ease when configuring individual components; (iii) configuration of the system as a whole and (iv) the possibility of managing different architectures making the execution of unity tests and integration of different architecture portions easier. Such aspects are very important for the implementation of reference.

As a result of this experience, many works are already being accomplished, such as (i) port of PUC-Rio's Ginga-NCL's presentation machine to the Common Core; (ii) the development of management tools for the architecture and conception of different middleware's versions; (iii) proposal of a conformance validation model for Digital TV's middleware.

9. Acknowledges

We thank the support of our institutions, the Laboratory of Digital Video Application of the Federal University of Paraiba and of the Federal University of Pernambuco, as well as the funding provided by Brazilian research agencies: National Education and Research Network (RNP) and Science and Technology Ministry (MCT) under the CTIC program[4].

10. References

[1] Peng, C. "Digital Television Applications" (PhD Thesis) – Helsinki University of Technology, Espoo, 2002.

[2] Morris, S. Smith-Chaigneau, A. Interactive TV Standards: A Guide to MHP, OCAP, and JavaTV. Focal Press, 2005.

[3] ABNT NBR 15606-2 Digital terrestrial television – Data coding and transmission specification for digital broadcasting – Part 2: Ginga-NCL for fixed and mobile receivers – XML application language for application coding, 2007.

[4] ABNT NBR 15606-5 Digital terrestrial television – Data coding and transmission specification for digital broadcasting Part 5: Ginga-NCL for portable receivers – XML application language for application coding, 2008.

[5] ABNT NBR 15606-4 Digital terrestrial television — Data coding and transmission specification for digital broadcasting Part 4: Ginga-J — The environment for the execution of

[4] CTIC Program: Available at: http://www.ctic.rnp.br/

procedural applications, 2010.

[6] Leite, L. E. C., et al. 2005. FlexTV – Towards a Middleware Architecture to Brazilian Digital TV System. Journal of Computer Engineering and Digital Systems. Vol. 2, pp. 29-50. 2005.

[7] Soares, L. F. G. 2006. MAESTRO: The Declarative Middleware Proposal for the SBTVD. Proceedings of the 4th European Interactive TV Conference (EUROITV 2006). Athens, 2006

[8] SBTVD. Brazilian Digital TV System Project. Available on: `http://sbtvd.cpqd.com.br`

[9] Souza Filho, G. L. de, Leite, L. E. C. e Batista, C. E. C. F. Ginga-J: The Procedural Middleware for the Brazilian Digital TV System. Journal of the Brazilian Computer Society. 2007, Vol. v12, pp. 47-56, 2007.

[10] Soares, L. F. G., Rodrigues, R. F. e Moreno, M. F.Ginga-NCL: the Declarative Environment of the Brazilian Digital TV System. Journal of the Brazilian Computer Society, Vol. v12, pp. 37-46, 2007.

[11] ITU J.200. ITU-T Recommendation J.200: Worldwide common core – Application environment for digital interactive television services, 2001.

[12] JavaDTV API. Java DTV API 1.3 Specification, Sun Microsystems, 2009. Available on: `http://www.oracle.com/technetwork/java/javatv/overview/index.html`

[13] Silva, L. D. N. et al. Digital TV Multiuser and Multidevices Application Development Support with Ginga. Amazonia Magazine. N. 12, pp. 75-84, 2007.

[14] ETSI TS 102 819: Globally Executable MHP (GEM). ETSI Standard May, 2004.

[15] Projeto PhoneME. Available on: `http://phoneme.dev.java.net/`

[16] Yaghmour , Karim . Building Embedded Linux Systems. O'Reilly Media, Inc, 2003.

[17] Miranda Filho, S. et al. Flexcm - A Component Model for Adaptive Embedded Systems. In: COMPSAC IEEE International Computer Software and Applications Conference, Beijing. p. 119-126, 2007.

[18] Caroca, C.; Tavares, T. A. Test Process Model to Ginga Common Core Components. In: Proceedings of the 15th Brazilian Symposium on Multimedia and the Web (WebMedia '09), Fortaleza, 2009.

[19] Gamma, E., Helm, R., Johnson, R. e Vlissides, J. Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, 1994.

[20] ABNT NBR 15603-2 (2008) - Digital terrestrial television — Multiplexing and service information (SI) Part 2: Data structure and definitions of basic information of SI. August, 2008.

[21] ISO 14496-20. Lightweight Application Scene Representation (LASeR) and Simple Aggregation Format (SAF), 2006.

[22] B24 Appendix 5 – Operational Guidelines for Implementing Extended Services for Mobile Receiving System, 2004.

[23] Cruz, V. M., Moreno, M. F., and Soares, L. F. Ginga- NCL: Reference implementation for portable devices. In Proceedings of the 14th Brazilian Symposium on Multimedia and the Web (WebMedia '08). ACM, New York, NY, 67-74, 2008.

[24] Moreno, F. M. A Declarative Middleware for Digital TV Systems. (Master Thesis); PUC-Rio, DI, 2006

[25] OCAP – Reference Implementation. Available on: `http://ocap-ri.dev.java.net.`

[26] Oliveira, M.; Cunha, P.R.F.; da Silva Santos, M.E.; Bezerra, J.C.C. Implementing home care application in Brazilian Digital TV. Global Information Infrastructure Symposium (GIIS '09), Hammamet, 2009.

[27] Trojahn, T.H.; Gonçalves, J.L.; Mattos, J.C.B.; Da Rosa, L.S.; Agostini, L.V. "A Media Processing Implementation Using Libvlc for the Ginga Middleware," Future Information Technology (FutureTech), In: In Proceedings of the 5th International Conference on Future Information Technology. 2010.

[28] Cabral, P. A et al. GingaCDN A Code Development Network to DTV Brazilian Middleware. In Proceedings of the 16th Brazilian Symposium on Multimedia and the Web (WebMedia '10). 1st Workshop of Interactive Digital TV. v2, Belo Horizonte, 2010.