A Methodology for Upgrading Legacy Middleware Ginga Implementations to Profile Ginga-D

Petrina Kimura, Bruno Farias, Jeferson Costa, and Romulo Fabrício
TPV Technology
Manaus, Brazil 69058–830
Email: {petrina.kimura,bruno.farias,jeferson.costa,romulo.fabrichio}@tpv-tech.com

Eddie B. de Lima Filho
TPV Technology
and Federal University of Amazonas
Manaus, Brazil 69058-830
Email: eddie.filho@tpv-tech.com

Abstract—Currently, the Brazilian digital television system is migrating its middleware approach from profile Ginga-C to an integrated broadcast-broadband (IBB) strategy, known as profile Ginga-D. Nonetheless, there is no indication regarding how to implement the new IBB services, while maintaining compatibility with older features. The present work proposes a methodology able to extending current implementations, in a systematic way, with the goal of harmonization maintenance among modules and conformance regarding current standards.

I. INTRODUCTION

All digital television (DTV) systems for terrestrial transmissions developed or adopted some form of interactivity based on a key element: a middleware. In summary, it is usually composed of an execution engine (e.g., Java virtual machine -JVM) and an associated API, the latter being necessary for accessing platform and DTV resources, in a transparent way. Initially, the digital video broadcasting (DVB) consortium created the multimedia home platform (MHP) [1], which was based on Java (JavaTV), the advanced television systems committee (ATSC) developed the digital TV applications software environment (DASE), which was later used as basis for the advanced common application platform (ACAP) [1], and the digital broadcasting experts group (DiBEG) developed and promoted a middleware standard for the integrated services digital broadcasting (ISDB), known as broadcast markup language (BML) [1]. The Brazilian digital television system (Sistema Brasileiro de Televisão Digital - SBTVD), in turn, adopted a hybrid middleware approach named as Ginga, which is composed of a declarative environment, based on nested context language (NCL) and Lua scripts, i.e., Ginga-NCL [2], and also a procedural one, based on Java, i.e., Ginga-J. Later, in 2016, Ginga-J [3] became optional, in order to favor Ginga-NCL, which was simpler, lighter, and royalty-free.

Currently, DTV systems around the world are already mature and have even been going through constant evolution. Nonetheless, the advent of the Internet provided a new level of content provision, in an environment apart from the broadcasting one. As a consequence, DTV system evolution shifted towards that, with the goal of connecting broadcast and broadband environments, which became known as integrated broadcast-broadband (IBB) [4], [5].

In that sense, DVB systems have started a migration from MHP to hybrid broadcast broadband TV (HbbTV) [6], ISDB has standardized HybridCast [7], and ATSC moved to ATSC 3.0 [8]. Ginga, in turn, deserves a more detailed explanation. In 2016, profile Ginga-C was standardized, which met most IBB requirements [9]; however, clear and robust support related to second-screen devices and streaming were still missing, which led to the development of profile Ginga-D [10], on late 2018, which is mainly based on a hypertext markup language revision 5 (HTML5) engine and a hypertext transfer protocol (HTTP) server with a representational state transfer (REST) application programming interface (API). Indeed, such an approach seems to be unique in the DTV middleware world and provides a flexible and easily extendable structure.

Although complete, the respective profile Ginga-D [10] does not indicate how to extend the previous one (Ginga-C) and a unified architecture is needed, mainly regarding communication among RESTful HTTP server, Ginga core (common core and NCL formatter) [2], and DTV platform, which raises a lot of questions and some degree of uncertainty.

The knowledge presented in the last paragraphs inspired this work, which proposes an architecture for profile Ginga-D and provides some specific methodologies for enabling standardized features. In addition, it also presents some experience acquired during the implementation of such an approach.

II. THE NEW GINGA PROFILE D

Traditionally, Ginga is composed by four main elements: NCL formatter, which is responsible for presenting NCL documents with embedded Lua scripts, common core, which provides access to platform and DTV resources, Lua engine, which handles and executes Lua scripts, and JVM, which is in charge of executing Java code [10].

Profile Ginga-A gather the basic middleware Ginga’s elements, which include image decoding, e.g., portable network graphics (PNG) and joint photographs experts group (JPEG); audio decoding, e.g., advanced audio coding (AAC) from standard moving picture experts group (MPEG) four with low complexity (LC) and MPEG-1 layers 1 and 2; character coding, e.g., unicode transformation format (UTF) eight and International Organization for Standardization (ISO) 8859-1;
Ginga CC WebServices, which provides the following APIs:

- Text formats, e.g., HTML and TXT; compressed Ginga applications; fonts of type open font format (OFF); and support to digital storage media command and control (DSM-CC), for receiving interactive applications. Profile Ginga-B, in addition to all preceding features, adds MPEG-1 video decoding. Finally, it is worth mentioning that middleware Ginga is usually delivered through a bunch of dynamic libraries.

Profile Ginga-C includes an application catalog user interface (AppCatUI) for application management; decoding of simultaneous video and audio content; increased application storage; application update; private bases, which are containers for Ginga-NCL applications; Lua security classes (e.g., cypher, digest, and signature); support to MPEG-4 containers (.mp4); HTTP protocol; and live editing commands (LECs).

Besides what was exposed, Ginga-NCL already provided presentation through secondary devices, with a region attribute called device, and remote content could already be retrieved for media nodes, through their src attributes (e.g., http://server_identifier/file_path/#fragment_identifier), which made it be considered as an IBB system [9]; however, a more structured access to DTV content by second-screen devices was still missing. Due to that, in 2018, NBR 15606 was revised and other volumes were added to it, i.e., 10 and 11. The former outlines a minimum profile for an HTML5 engine, while the latter defines an HTTP server with a REST API, which is called Ginga common core (CC) web services (Ginga CC WebServices). In addition, some important features previously available only for Ginga-J and other imperative environments were identified and added to Ginga-NCL, which became known as Gaps. In summary, profile Ginga-D is composed by profile Ginga-C, Gaps, HTML5 engine, and Ginga CC WebServices [10]. Moreover, profile Ginga-D added other specific features, such as video H.265: audio coding 3 (AC-3), enhanced AC-3 (E-AC-3), and AAC LC high efficiency (HE) versions 1 and 2; subtitles internet media subtitles and captions 1.0.1 (IMSC1) timed text markup language (TTML) and web video text tracks (WebVTT); MPEG dynamic adaptive streaming over HTTP (MPEG-DASH) and HTTP live streaming (HLS); and MPEG common encryption format (MPEG-CENC). From the point of view of developers, interactive applications can now be written in NCL or HTML5, while any external element (e.g., second-screen device) or internal one (e.g., smart TV, native, and Ginga applications) can access resources and control DTV receivers, through Ginga CC WebServices, which provides the following APIs:

- Application authorization: pairing (via quick response - QR code or identification numbers) and token issuing;
- DTV context access: acquisition of service list, channel change, and service components control;
- Communication with Ginga environment: access to application list, application control, acquisition of application properties (which can also be changed) and files, stream event listener, and LECs;
- Access to PSI/SI data: retrieval of sections of time offset table (TOT), network information table (NIT), service description table (SDT), program association table (PAT), bouquet association table (BAT), program map table (PMT), and electronic program guide (EPG) information;
- Broadcast content access: access to streams and media players, while streaming is performed from DTV receivers, according to real time streaming protocol (RTSP) and real time protocol (RTP), and from external servers, according to protocols DASH and HLS;
- Platform integration: platform capabilities, digital rights management (DRM), deep link management, and application execution (installed in DTV receivers);
- Security: creation of links between a DTV service context and clients, through the sending of a bind token by broadcasters, which is managed by receivers.

Consequently, such a new scenario should be developed, integrated with the former profile Ginga-B/C, ported to a commercial DTV platform, and then tested and evaluated, in order to provide conformance with current standards. Indeed, Ginga does not provide a logo acquisition procedure, but an official test suite [10] is provided for a self-certification process to be carried out by each manufacturer. This way, testing approaches must also be developed, in order to guarantee correct and suitable implementations of profile Ginga-D. Finally, only a high-level description of profile Ginga-D is available, without any clue regarding implementation and testing strategies, which is tackled by the present paper.

### III. The Proposed Methodology

When talking about middleware Ginga’s modules, it is important to mention that most manufacturers will start developing over their previous versions, given that profile Ginga-D builds on top of them; however, given the huge amount of new interfaces, algorithms, and protocols, in addition to backward compatibility with legacy applications and systems, a more refined architecture is necessary.

In the present case, the base Ginga implementation provided only profile Ginga-B, which also led to development and integration of profile Ginga-C. Firstly, regarding HTML5 applications, those can be sent as medias in NCL documents, stand-alone Ginga applications, or even be native elements. Indeed, given that an HTML5 engine is a very flexible environment and has the potential to provide HTML5-centered platforms, i.e., everything could be provided in that format, a care design is needed. As a self-contained element, it can be provided as an independent library, which can be accessed by Ginga through an specific access layer. This way, possible bugs and errors are left isolated, which is better managed than if it was an integral part of the middleware itself. This way, an instance is created through the mentioned access layer, which provides a suitable execution environment. Besides, it is worth mentioning that in an HTML5-centered platform, it would be interesting to have a way of accessing DTV resources, in such a way that even a zapper module, i.e., the application in charge of controlling a DTV receiver, could be developed. As a result, a JavaScript API was designed, which is based on the following enhanced APIs inspired by HbbTV [6]: Object
Factory, Broadcasting Control, Application Manager, and Channel Config, in addition to a newly developed API focused on program specific information (PSI), service information (SI), and transport packet (TP) retrieval [11], which is called DTV Java-script (JS) API. Finally, abstractions are needed for accessing media infrastructure, input management, and graphics back-ends, which is shown in Fig. 1.

Fig. 1. The proposed architecture for the HTML5-engine.

Gaps mainly provide features initially available only in imperative environments, such as character coding; transmission control protocol (TCP) and user datagram protocol (UDP) communication; ZIP file processing; pixel-buffer dumping and coded-image loading, in JPEG and PNG formats; HTTP/HTTP secure (HTTPS) requests; presentation of DRM content, i.e., class mediakeysession in Lua [10], with access to content decryption modules; stream injection through Lua scripts; and service selection. As one may notice, such services are usually available in underlying systems or provided through open-source code. As a consequence, they might not be directly implemented in Ginga, but instead accessed through suitable interfaces for Lua scripts, i.e., they only initially handled by Lua and then translated and handed over to specific managers, through abstraction layers. One example of that is the ZIP feature, which can be implemented through minizip [12]. The strategy for Gaps is shown in Fig. 2.

Fig. 2. The proposed architecture for Gaps.

Profile Ginga-C involves enhancements mainly related application management. For instance, AppCatUI handles applications transmitted through the DTV signal, delivered through the broadband interface, installed, and resident. NCL applications are managed through specific containers, which are known as private bases [10], while HTML5 ones present no constraint. When a channel is tuned, a root private base is created, with an identifier (baseId) given by the value of network_id and found in PMT [13], which is able to encapsulate an application, as is the case in one-seg signals [10]. In addition, when a specific service is tuned, which happens to fixed receivers, another private base for the respective application that is nested inside the first one is then built. That kind of organization is very interesting, because LECs sent to a root private base may be forward to its children, which is a feature of the proposed methodology that prevents errors in baseId information, provides robustness, and causes no damage, given that only one private base is active [10].

LEC's allow on-the-fly modification of source code, without downloading a new DSM-CC object carousel for updating a given application. An example is given Fig. 3, where a new region, i.e., a screen portion where media content can be rendered, is added to an application. LECs are sent as stream events [10], in TPs tagged with a specific stream_type (0xC). With a brief analysis, one may notice that integration of profile Ginga-C results in modifications in both common core and NCL formatter, in order to provide the required services. Based on that, the proposed methodology splits LEC handling into structures placed into both NCL formatter and common core, which depends on a specific command’s goal: if it is related to storage, such as saveBase and saveApplication [10], it is handled by both NCL formatter and common core; otherwise, it is handled only by the former. Moreover, private bases were integrated into NCL formatter, and Lua modules were included into the Lua stack. Finally, AppCatUI is not exactly a module inside Ginga, but instead manages applications through calls to the latter, in order to retrieve, save, and delete elements. Indeed, as NBR 15606 states that AppCatUI “depends only on a manufacturer”, it is, in the proposed methodology, a stand-alone module that can even be implemented over the HTML5 engine, which accesses common core and NCL formatter, as shown in Fig. 4 (profile Ginga-C’s elements are in orange).

Ginga CC WebServices provides a REST API accessible through HTTP/HTTPS, whose main goal is to allow local and
external applications to explore resources from DTV receivers. In order to perform that, it must access the middleware Ginga or the underlying platform itself (e.g., platform capabilities), depending on each specific call. Nonetheless, standard NBR 15606 [10] defines that Ginga CC WebServices can also offer streaming through MPEG-DASH or HLS and respond to requests when no DTV channel is tuned or the DTV environment is not loaded, which indicates a life cycle apart from that of the main middleware’s core, i.e., it is alive even when a smart environment is in use. In addition, announcements are performed with the simple service discovery protocol (SSDP), where access information is multicast in the local network. As a consequence, in the proposed methodology, Ginga CC WebServices was designed as a stand-alone module, which communicates with common core, NCL formatter, and underlying platform, as shown in Fig. 5.

Fig. 4. The proposed architecture for profile C.

As mentioned earlier, any Ginga profile D implementation needs to be in conformance with current standards. Given that SBTVD provides an official test suite and adopts the self-certification strategy, each manufacturer must ensure that its middleware Ginga implementation is correct. Due to that, in the proposed methodology, an automated test equipment (ATE) sends test applications for receiver evaluation. In order to check the complete chain, all necessary PSI/SI information is created, multiplexed with a test application, and included in a standard transport stream (TS), which is then sent through a standardized ISDB-T interface, as defined by SBTVD. ATE runs on a personal computer and its execution control is able to monitor application execution (via serial port) and then evaluate results, in order to check conformance. Besides, it emulates a second-screen device by performing requests to Ginga CC WebServices, with suitable clients/servers (TCP, UDP, HTTP, etc.). There is a webcam for capturing video and also QR codes, in order to enable authorization for accessing services and evaluate visual outputs, and a microphone, which is used for verifying audio output.

An interesting aspect that should be discussed is how successful the proposed methodology is regarding middleware extension and integration. In that sense, three key figures are of paramount importance: percentage of reused legacy code, development time, and error correction related to legacy features. Indeed, the application of the approach presented here is in its initial phase, regarding the development of profile Ginga-D; however, we were able to notice shorter development periods, which includes known competitors that are also part of the SBTVD’s standardization committee and what was spent before, and a percentage of code reuse higher than 95%. Moreover, overall conformance has been kept thus far, with low regression rate, after code integration, and legacy feature preservation.

Finally, the feasibility of the proposed approach is being accessed during the on-going coding task, as one may notice from the last paragraph, which has been considered successful, given test results reported after each development phase.

Fig. 5. The proposed architecture for Ginga CC WebServices.

IV. CONCLUSION

This work proposed a methodology for implementing profile Ginga-D, while starting from a traditional middleware Ginga implementation conforming to profile Ginga-B. It employs a clever architecture mainly based on abstraction and access layers, with the goal of decoupling elements, favoring maintenance, and providing robust and flexible implementations.

Indeed, the proposed architecture includes a non-standardized JavaScript API, named as DTV JS API, which enables the development of native HTML5 applications capable of controlling a commercial DTV receiver, in such a way that every module or application can be developed for running over the same engine (HTML5-centered receiver). Moreover, Ginga CC WebServices is a stand-alone module, as also happens to AppCatUI, which can respond to requests when DTV is not in use. Consequently, it can be employed by smart applications or even invoke them, if there is a registered deep link [10].

Finally, the proposed methodology devises a systematic way for developing profile Ginga-D implementations, which are more robust, present easier maintenance, and create a more compliant DTV stack, due to the use of ATE.

REFERENCES


