Extended features for the Ginga-NCL environment
Introducing the LuaTV API

Rafael Rossi de Mello Brandão, Guido Lemos de Souza Filho
Departamento de Informática
UFPB
Paraíba, Brasil
{rafael,guido}@lavid.ufpb.br

Carlos Eduardo Coelho Freire Batista, Luiz Fernando Gomes Soares
Departamento de Informática
PUC-Rio
Rio de Janeiro, Brasil
{cbatista, lfgs}@inf.puc-rio.br

©2010 IEEE. Personal use of this material is permitted. However, permission to reprint/republish this material for advertising or promotional purposes or for creating new collective works for resale or redistribution to servers or lists, or to reuse any copyrighted component of this work in other works must be obtained from the IEEE. The original publication is available at http://dx.doi.org/10.1109/ICCCN.2010.5560066
Extended features for the Ginga-NCL environment
Introducing the LuaTV API

Rafael Rossi de Mello Brandão, Guido Lemos de Souza Filho
Departamento de Informática
UFPB
Paraíba, Brasil
{rafael, guido}@lavid.ufpb.br

Carlos Eduardo Coelho Freire Batista, Luiz Fernando Gomes Soares
Departamento de Informática
PUC-Rio
Rio de Janeiro, Brasil
{cbatista, lfgs}@inf.puc-rio.br

Abstract — The Ginga-NCL environment is responsible for the presentation of declarative (NCL) applications on the Brazilian Digital TV System (SBTVD), and it is also the ITU-T H.761 Recommendation for IPTV middleware. Lua is the scripting language of NCL. ITU-T H. 760 series also defines two sets of NCLua API for the development of IPTV applications: the Core and the Extended API. The Core comprises the basic NCLua API, which are part of the Ginga-NCL original specification for the Brazilian terrestrial DTV system. The Extended aims to provide other relevant functionalities incorporating some of the features commonly present on other imperative environments. LuaTV is part of the draft specification for the NCLua Extended API. LuaTV features are divided in four functional categories: metadata, with functionalities related to the accessing Digital TV metadata information; security, providing mechanisms for encrypting and authenticating data; HAN, offering high-level access to resources commonly available on home networks; and widget, aimed at graphical support to applications. This paper presents the current LuaTV specification and its implementation.

Keywords—Digital TV; Digital TV middleware; Ginga; NCL; Lua

I. INTRODUCTION

During the researches for the development of the Brazilian Digital TV Systems, an innovative middleware specification was conceived, bringing new possibilities for the development of Digital TV applications. The Ginga middleware offered many features that were unknown for the current specifications. The Ginga architecture comprises two application execution environments: a declarative, for the presentation of NCL applications [1]; and an imperative, aimed at controlling the execution of entities written in some imperative language, for example Java in the case of the Brazilian terrestrial DTV [2].

The Ginga-NCL environment supports application written using the NCL language that has Lua as its scripting language. Ginga-NCL includes the NCLua Core standard API with relevant functionalities for the Digital TV applications.

Lua TV API is intended to complement the current Ginga-NCL specification, allowing users to develop NCL applications with extended support for multi-user applications (through the use of multiple networked devices), and adding mechanisms for metadata access, secure communication and user interface development. Thus, the current version of the API comprises four functional categories: Metadata, gathering functionalities related to the access of the metadata information contained on Digital TV transmissions; Security, providing mechanisms for data encryption and authentication; HAN, offering high-level access to services available at home area networks; and widget, providing graphical components for the development of widget [6] fashioned applications.

This paper presents the LuaTV API using the following structure: the next section brings relevant information regarding the Ginga-NCL environment, so that the LuaTV context can be well understood; the third section discusses similar works, with a critical view on their design decisions; the fourth section comprises the LuaTV architecture and discusses the features provided by the API; the fifth section is composed by technical information regarding its current implementation; and finally the last section concludes with some possibilities for further developments.

II. GINGA-NCL AND LUA

Ginga-NCL is the Ginga subsystem in charge of the presentation of declarative documents, written in NCL [1]. NCL is based on the XML language, with a broader focus than other declarative languages with similar purposes such as XHTML-based middlewares. The space-time synchronization defined by NCL connector and link elements; the adaptability defined by switch and descriptor elements; the support to multiple devices, and the support to live application production are the key features of NCL [1].

A NCL document defines how media objects are structured and related in time and space. As a glue language, NCL doesn’t restrict or specify the media objects’ contents. Imperative objects can be inserted in NCL documents to bring additional computing capacities to declarative documents. Ginga-NCL supports at least one type of imperative object in a NCL document: Lua imperative scripts named NCLua media objects.
NCLua media objects rely on the facilities of the standard library Lua API, part of the Ginga-NCL NCLua API. The NCLua API provides four modules: the canvas module for drawing graphics primitives, event module for communication between NCLua objects and other components (such as the NCL Formatter) through events, the settings module where variables defined in a NCL document can be retrieved and assigned and the persistent module for definition of persistent variables which can be used by other imperative objects.

III. RELATED WORKS

This section presents some works related to the different features of the introduced API. The next sections discuss works that implement solutions related to the functional packages of the LuaTV API: Graphical User Interface (GUI), device integration, security and retrieval of television services metadata.

A. User Interface API for Digital TV

LuaOnTV [7] is a framework for input and output user data in graphical components for interactive applications; it is implemented in an object-oriented approach. This framework was developed at the ENE-UnB DTV laboratory, under GPL license, in order to facilitate the development of NCLua applications so that the NCL could be responsible only for media synchronization [8]. Its architecture is based on Java GUI API model, which is a little out of sync with Lua language focus, which is to provide simple mechanisms for development and extension, not defining a programming model and an API specification with a great number of facilities and all the possible uses of it [9].

The HAVi standard defines a set of Java API for graphical interfaces known as HAVi Level 2 GUI, including a set of widgets that do not require a windowing system and also a set of classes for management of scarce resources, to enable applications to share the screen without a window manager. The org.havi.ui package provides functionalities equivalent to the Java AWT (Advanced Windowing Toolkit), the Java graphic API. The HAVi GUI API use platform independent aspects of the java.awt package extending its specification with remote control management and graphical system information retrieval. HAVi Level 2 GUI is used by DVB-MHP [16] and its compatible middlewares (GEM – Globally Executable MHP).

Ginga-J (JavaDTV API) user interface API was specified to be a functional equivalent to the GEM API, and it has two graphic packages: a widget API, named LWUIT (Light-Weight User Interface Toolkit) and a DTV UI API. The com.sun.dev.lwuit is based on the same component/container composition with similar design and terminology as Java AWT/Swing while com.sun.dev.ui supports specific TV functionalities. DTV UI offers, among other things, I/O devices support (keyboard, mouse and remote control) and also provides screen and graphical planes (video, background, closed captions, etc.) abstractions. Each of these planes has an associated container where components can be inserted and have their layout managed.

B. Remote Devices Integration

The declarative environment Ginga-NCL defines two communication models with secondary display devices for distributed application presentation [10]. In the first model (for passive devices) the same content is displayed on a single navigation control; the second model (for active devices) allows devices to control the content presentation independently with a completely individualized experience in each connected device. The set of registered devices on a base device (unique) defining an application domain, as the base device manages the application presentation using the registered resources. The base device shall transmit ready-to-be-displayed audio and video samples to passive devices, so that all devices always show the same content. For the active devices class the base device transmits parts of the application, which are media objects (including NCL code) to be presented and locally controlled.

Ginga-I InteractionDevicess API is present in the SBTVD specific API package (br.org.sbtvd.interactiondevices) and was proposed during the Ginga-J environment specification [11]. The API uses the STB’s computing power so that all the user interaction on device is processed in the STB itself, and only the result of this processing is sent to remote device. The API provides methods to retrieve information about registered devices (device type, available features and resources etc) and also to explore the features available like audio and video recording, still images, and so on. Each device has a graphic container (DTVContainer in JavaDTV) used to interface composition that will be displayed transparently on device’s screen. Ginga-J only offers the option for communication with each device independently, without the device class abstraction used by Ginga-NCL for distributed presentation, which we understand that is more appropriated for the declarative environment that LuaTV aims to extend.

The ARIB (Application Execution Engine Platform for Digital Broadcasting) STD-B23 specification defines an application execution environment for DTV adherent to the ITU J.202 recommendations (analogue to the Ginga-J imperative environment); the specification also defines a specific Java API package for the ISDB system. The jp.or.arib.tv.peripheral [12] is the core package for the integration device in the specific ARIB API. This package provides classes and interfaces for device discovery, device registering, property and features retrieval, device states retrieval and also reading and writing for communication. This device integration API is conceptually similar to the SBTVD InteractionDevices.

C. Security

The Java Security API [13] (java.security package) defines a set of API together with implementations of commonly used security algorithms, protocols and mechanisms. The API covers many issues regarding security including encryption, public key infrastructure, secure communication, authentication and access control. Three classes are particularly relevant to the LuaTV API: MessageDigest, Signature and Cipher classes. These classes together provide basic security features, which are covered by LuaTV providing facilities to data verification, key creation and verification and data encryption.
Lua MD5 [14] is an open library that provides basic cryptographic facilities for Lua 5.1. Two modules are present in this library: the *md5* module and the *des56* module offering facilities to digest (hash), encryption/decryption using MD5 and DES algorithm. The library’s design, as well as almost all the other standard Lua libraries, is quite simple and straightforward. The LuaTV specification took into account this simple design and easy utilization paradigm applied in Lua libraries specification.

D. Service Information

Service Information (SI) API commonly provides access to the data tables which are multiplexed within a MPEG-2 Transport Stream [15]. Such information is related to the service being transmitted, including audio and video streams identifiers and parameters, and also the textual description of its contents. NCLua Core API offers the possibility of accessing specific SI data through the NCL settings node - where a set of SI tables are hierarchically represented and by which their particular fields are able to be accessed. LuaTV approach aims to provide a generic mechanism for accessing SI information not related to any particular system, so the following paragraph presents a discussion on the approaches considered for the modeling of a generic SI API for the LuaTV specification.

The Ginga-J middleware offers access to those tables through the JavaTV Service API (package *com.sun.dtv.service*). A single class is defined by this package, the *SITable*, to be used for a generic access to the SI Tables, using platform dependent references. The DVB SI [16] is used by the MHP middleware to provide access to DVB system specific information, thus reflecting the SI tables used by such. The access is provided through synchronous and asynchronous calls (using events and lists), depending upon what kind of information is being retrieved (the synchronous calls returns pre-cached information). The ARIB SI used by the ISDB system is based on the SI, with some adjustments to its particular system tables and features.

IV. THE LUATV API

This section presents the LuaTV API focusing on its architecture, module features and design, and integration with the Ginga middleware.

A. Overview

Essentially, NCLua objects bring extra capabilities to NCL documents offering imperative facilities to the document author. LuaTV provides additional features to the NCLua API and also provides abstraction of other software layers present on Ginga middleware specification. The API is located in the Ginga specific service layer and currently has four independent API packages: *Widget, HAN, Metadata* and *Security*. This first version of the LuaTV API specification focuses on some features available to imperative environments such as Ginga-J and DVB-MHP [16] that were missing in Ginga-NCL NCLua API. Figure 1 shows the LuaTV integration to the Ginga middleware architecture (its reference implementation).

B. Architecture

LuaTV’s packages are functionally independent and follow the same minimalist and low-level model of the NCLua Core API specification. LuaTV needs to communicate with Ginga’s common core components to offer some functionalities of the API; NCLua API is also used to display graphic information and to exchange information with the NCL Formatter through the *canvas* and *event* module.

The LuaTV Widget API doesn’t offer a complete widget collection in an object oriented fashion as it is normally done by graphical toolkits. Instead, it aims to provide simple mechanisms so user can build arbitrary visual components focusing mainly on user input and output, as NCL documents already offer a range of mechanisms for interface compositions, but lacks on interactive elements. There are three modules present in this package: *window, cursor* and *textarea*.

The *window* module is responsible for filling a given canvas (commonly associated with a NCL region) with content retrieved from a local or remote URL. The *cursor* module supports pointer features (associated to an image) to be displayed on a *canvas*. The *textarea* module provides mechanisms to capture inputted user text to be displayed on a *canvas* too. Figure 2 illustrates the architecture of the Widget API.
The LuaTV HAN package provides facilities to the discovery and utilization of remote devices connected to the receiver (or base device) and its resources. These resources can be used to provide audio and video streams or binary data (e.g. text or key events) to a NCL document. There are two modules in this package: devicemanager and deviceservice.

The devicemanager module is responsible for the dynamic registration of device classes and for providing access to the metadata describing the device classes and its device members. It is also responsible for the notification of the association of devices on the available classes – events of the class "device", types "join_class" and "leave_class". The events carry the identification of the class and the identification of the device.

Through the devicemanager module it is possible to instantiate the deviceservice module. The deviceservice module allows generic access to the services provided by the secondary devices belonging to a device class. The device service module provides a generic interface for the submission of service requests, and also a mechanism for data reception from such services. The data reception is done through the reception of events of the class "device", type "data", carrying the device class identification, the device's identification, the device service's identification, the related data and the request's identification as well. Figure 3 shows the HAN package internal architecture.

The Device Integration component in the Ginga’s common core layer (shown in Figure 3) is not part of the proposed architecture of the middleware specification [17]. The development of this component, though, has become critical to the reference implementation of the API, since that it relies upon platform specific features, and the design of LuaTV aims at a more abstract set of functionalities. It is important to notice that, due to security constraints, the communication with remote devices may be limited for the imperative environment; hence, its components may only be available for the development of resident applications.

The Metadata API provides a mechanism for retrieving generic Service Information (SI) in a DTV broadcast stream. Information which are present in Transport Stream tables are queried and received by handlers (or listeners) registered in the metadata object in an asynchronous scheme. This API was modeled aiming a generic approach so that implementations of this package could handle different tables and formats among the many DTV systems. This package has a unique module where handlers can be registered to recover the data in question. It is possible to recover an entire SI table (conveniently represented as a Lua table) or just a specific field of a table. Figure 4 shows Metadata package architecture; noticeable is the fact that a communication with the SI common core component is necessary for this API.

The author in the Ginga-NCL environment does not address aspects regarding security in a NCL document. Instead, it is assumed that media players must support the eventually required security protocols. This leaves the document author tied down concerning security control for his application. The LuaTV Security API provides security facilities for applications offering generation and verification of digital signatures, message digest generation, and encryption for secure data transfer. Applications like T-Bank, T-Commerce, or public utilities [11], or a polling application are some examples of applications that need to be in charge of security control.

The LuaTV Security API has three modules: signature, digest and cipher. A signature object provides methods for digital signature generation and verification. The digest module offers facilities for message digest generation. Data encryption and decryption is provided by the cipher object. All the security algorithms used by these modules are defined by a string argument in constructors; the available set of algorithms
is dependent of the API implementation. Figure 5 presents the security package architecture with Lua md5 library.

![LuaTV Security](image)

**Figure 5.** LuaTV Security package

V. CURRENT IMPLEMENTATION

An Open Source implementation of the presented Ginga architecture is been developed at the Digital Video Applications Lab (LAVID\(^1\)). In 2009, the GingaCDN (Ginga Code Development Network) project started with distributed and collaborative software development purposes, thus the OpenGinga considers the Ginga-NCL reference implementation\(^2\) componentization and the FlexCM component model [18] allowing a modular development through an interface and component connection scheme. Common core components in OpenGinga can be defined, implemented and tested as a stand-alone solution and later connected to other components through the FlexCM execution environment. The LuaTV API reference implementation is under development taking advantage of the OpenGinga environment as this Ginga reference implementation provides a completely modular framework with architectural extension facilities.

GUI and Security packages are developed completely in Lua language using the NCLua API and Lua standard libraries. In contrast Metadata and HAN packages needed Ginga's common core functionalities, specially the HAN package that required a new component specification, which suits both Ginga-J and Ginga-NCL (and thus LuaTV) specification.

The common core layer is completely implemented in C/C++ language. As an extension language Lua was designed to exchange data with other languages. In this case, an abstract stack allows the communication with C/C++ code.

As mentioned, the Device Integration component architecture was designed not only to support the HAN API, but any other device integration API like the one present in SBTVD specific Ginga-J API. The communication between the two layers is accomplished through the JNI (Java Native Interface) programming framework which allows native code communication by the Java Virtual Machine. Figure 6 shows the integration of the conceived component architecture with other Ginga specification layers.

The following functional entities were defined for the Device Integration architecture:

- Event Service: Provides an event registration service. Events are generated by the middleware and can encapsulate binary data objects (like files, key press events) or reference to a media stream.
- Device Session Mediator: Defines communication sessions with devices or groups of devices (multicast communication). The component is responsible for filtering incoming events from Event Service and sends them to the appropriate sections. This association is done according to the capabilities of the devices associated with the event.
- Resource Proxy: This is an interface for device's resources (objects and media streams). It uses a standard semantic from Salutation Service to identify which resources will be used for device or group communication.
- Salutation Service: Allows device and associated resources registering using a standard semantic (such as used by UPnP services)
- Device Monitor: Provides device retrieving, connection states verification and allows data transfer.

Currently the LuaTV implementation is already functional. The implementation also defines a set of applications that may explore its functionalities representatively. Bellow the execution of one of these applications over the current LuaTV implementation: the LuaTV Quiz (shown on Figure 7 and 8). In this application some questions are displayed on connected devices’ screen and the participants use these remote devices to answer the questions. A scoreboard is shown on TV with the updated rank of total correct answers of all players.

---

1. http://www.lavid.ufpb.br
2. http://www.gingancl.org.br
A reference implementation will be made available, considering the ever-changing Digital TV scenarios for the development of multiuser and multidevice applications using the Ginga middleware. The implementation is being matured, and this may affect the actual specification, based on the tests results and also considering the ever-changing Digital TV scenario. The reference implementation will be made available as part of the results of the Ginga-CDN project, using the MIT license as Lua language does.

REFERENCES


