Ginga-NCL: Declarative Middleware for Multimedia IPTV Services

Luiz Fernando Gomes Soares, Marcio Ferreira Moreno, Carlos de Salles Soares Neto, and Marcelo Ferreira Moreno, Pontifical Catholic University of Rio de Janeiro

Ginga-NCL: Declarative Middleware for Multimedia IPTV Services

Luiz Fernando Gomes Soares, Marcio Ferreira Moreno, Carlos de Salles Soares Neto, and Marcelo Ferreira Moreno, Pontifical Catholic University of Rio de Janeiro

ABSTRACT

This article presents the innovative features of Ginga-NCL, an open middleware specification for multimedia IPTV services. Ginga-NCL relies on the Nested Context Language, a domain-specific declarative language targeting multimedia application authoring. As a glue language, NCL relates media objects in time and space without restricting or imposing any media content type, including media objects with imperative and declarative code written using other languages. Other NCL features include support for multidevice presentations, content adaptations, presentation adaptations, and advanced code reuse. Ginga-NCL allows NCL applications to be modified on the fly by means of live editing commands. Initially defined as the standard middleware for the Brazilian terrestrial DTV system, Ginga-NCL has recently become part of ISDB standards and an ITU-T Recommendation for IPTV services.

INTRODUCTION

IPTV systems have brought the last triple-play service to IP networks. This digital convergence should be supported not only at low-level network layers, but also at the application layer. Thus, an application programming language that is able to support, integrate, and coordinate a number of different media objects in a simple and synchronized manner is a very important requirement in this new domain. The solution can come from languages using the declarative paradigm.

Declarative languages emphasize the high-level description of an application, rather than its decomposition into an algorithmic implementation, as is done when using imperative languages. Such declarative descriptions are easier to devise and understand than imperative ones, which usually require a programming expert. However, declarative languages typically target an application domain and define a specific model to design applications in this domain. When an application design matches the declarative language model, the declarative paradigm is generally the best choice.

Nested Context Language (NCL) [1] is a declarative XML-based language initially designed for hypermedia document specification for the web. The language's flexibility, reuse facility, multidevice support, application content, presentation adaptability, and mainly its intrinsic ability to easily define spatiotemporal synchronization among media assets, including viewer interactions, make it an outstanding solution for all kinds of digital TV (DTV) systems, particularly IPTV systems. For particular cases, for example, when dynamic content generation is needed, NCL provides Lua (imperative) scripting language [2] support.

In 2007 NCL was adopted in the Brazilian terrestrial DTV standard, SBTVD. In the beginning of 2009 NCL and its user agent, called Ginga-NCL, became part of International Standard for Digital Broadcasting (ISDB) standards (the previously known Japanese standard now increased with Brazilian improvements) and part of International Telecommunication Union — Radiocommunication Standardization Sector (ITU-R) Recommendation BT 1699. Also in 2009, NCL and Ginga-NCL became an ITU — Telecommunication Standardization Sector (ITU-T) Recommendation for IPTV services [3]. NCL and Ginga-NCL were designed at the TeleMídia Laboratory at the Pontifical Catholic University of Rio de Janeiro (PUC-Rio). The work was coordinated by the authors of this article, who also chaired ITU-T H.761 and the Brazilian DTV Middleware Working Group. Independent of the distribution network where they may be applied, all Ginga-NCL and NCL specifications are open and totally royalty-free.

A reference open source implementation of Ginga-NCL[1] has been developed to easily integrate a variety of media-object players (for audio, video, image, text, etc.), including imperative execution engines and other declarative presentation engines. As aforementioned, Lua code can be embedded in an NCL application by means of a special NCL object type called NCLua. Because of its simplicity, efficiency, and powerful data description syntax, Lua was considered the natural scripting language for Ginga-NCL. The Lua engine[2] is small and written in ANSI/C, making it easily portable to several hardware platforms. Lua is one of the most popular languages in the entertainment area.

This article presents some NCL features and discusses how Ginga-NCL can be used in IPTV

[1] *Ginga-NCL reference implementation is available under the GPLv2 license at http://www.ncl.org.br*

[2] *The Lua engine is also distributed as free software under the MIT license.*

0163-6804/10/$25.00 © 2010 IEEE

systems. The article is organized as follows. The next section overviews some related work. We then introduce the main NCL concepts. We present the Ginga-NCL architecture for IPTV systems. The last section is dedicated to our final remarks.

Related Work

DTV applications can be partitioned into a set of declarative applications and a set of imperative applications [4]. A declarative application is an application whose initial entity is of a declarative content type. An imperative application is an application whose initial entity is of an imperative content type.

Most terrestrial DTV systems offer support for both paradigms. For example, the European Digital Video Broadcast (DVB)[5] system and the American Advanced Television Systems Committee (ATSC) [6] system support Java, EcmaScript, and XHTML in their middleware specifications; the Association of Radio Industries and Businesses (ARIB) Japanese system also provides both declarative and imperative support, although current products implement only the declarative language BML [7], with EcmaScript as the scripting language. The Brazilian SBTVD system specifies Java and Lua as imperative languages for its middleware, called Ginga. NCL is the declarative language of Ginga.

Besides offering an application programming interface (API) for interactive applications like terrestrial DTV, an IPTV middleware integrates specific IPTV components and services. Among those components, the most common are video on demand (VoD), linear TV, IP communication, remote software update, and content search engines.

Most IPTV service providers use proprietary middleware implementations [8] from vendors like Microsoft, Minerva, Orca, Siemens, SoftAtHome, and Alcatel-Lucent. Microsoft Mediaroom (an evolution of Microsoft TV) [9] is based on the .NET framework compact for end terminals, in order to offer a C#/XML API (Silverlight) for applications. Minerva Networks has developed a middleware known as iTVManager [10], which offers a C language interface for its applications. The RIGHTV middleware [11], developed by Orca Interactive, is based on XHTML browsers. Siemens has acquired Myrio [12], which offers Java and XHTML APIs. The SoftAtHome operating platform [13] is a recent initiative from Orange, Thomson, and Sagem to provide a declarative application environment based on HTML, JavaScript, and Adobe Flash. Alcatel-Lucent's MiViewTV [14] offers a development framework totally based on Adobe Flash.

Lightweight Interactive Multimedia Environment (LIME) [15], like NCL, is an ITU-T Recommendation for IPTV multimedia applications and is based on the Japanese BML standard for broadcasting. LIME specifies a small profile of XHTML, CSS, DOM, and EcmaScript, where broadcast-specific functions from BML were removed. The LIME EcmaScript API contains extensions to deal with IPTV-specific functionalities such as IPTV EPG, persistency, content licensing, media presentation control, and TCP/IP communications. LIME supports not only user interaction events, but also broadcaster events, by means of *bevent* and *beitem* elements. A *beitem* element describes an event message and the corresponding action (usually a function written in EcmaScript) to be accomplished when the event occurs. Note that although spatiotemporal synchronization among media objects can be controlled by broadcast systems, the behavior must be imperatively defined using scripts.

After the advent of broadband TV, broadcast content may share its audience with downloaded widgets and online tasks like browsing and messaging. This has pushed broadcasters and service operators to define standardized platforms that harmonize and explore this hybrid scenario.

HbbTV [16] supports web technologies like XHTML and EcmaScript for broadcast or downloaded applications. However, application signaling is done in the broadcast domain. The HbbTV standardization process started in the third quarter of 2009 in the European Telecommunications Standards Institute (ETSI).

The BBC's Project Canvas [17] is an initiative to standardize a common platform that broadcasters can join, making their multimedia content available to compatible IPTV terminals. Some discussions and consultations are still running at BBC Trust, and there are no technical standards published yet, although web technologies are said to be a natural choice.

Verizon's FiOS TV [18] offers a framework for developing TV widgets based on the Lua language. The framework incorporates the Lua virtual machine and extended Lua APIs such as Graphics, Events, Timers, and Webservices.

Authoring DTV applications using imperative languages is more complex and more error-prone than using declarative domain-specific languages (DSLs). Moreover, imperative interpreted system languages like Java have engines that are very resource consuming and have a considerable memory footprint. On the other hand, XHTML carries a legacy from previous technologies developed for text navigation and formatting, and lots of add-ons have been created to overcome its limitations. XHTML is focused on user interactions for media asset presentation synchronization, forcing application authors to solve spatiotemporal synchronization that goes further than simple user interactions, as well as content and presentation adaptations and other issues, using imperative objects, usually implemented using EcmaScript. Thus, the great advantage of using declarative DSL is lost, with the expense of using a scripting language greedy in CPU and memory consumption.

A declarative approach that fulfills the main requirements of a DTV application, relegating to the imperative approach only particular computations, seems to be the right solution for a DTV middleware API. This approach would boost integration, simplicity and better resource usage in IPTV platforms. Besides that, it would make the authoring process easier and less error-prone.

NCL [1] and MPEG-4 Lightweight Application Scene Representation (LASeR) [19] are the technologies currently closest to fulfilling these

A declarative application is an application whose initial entity is of a declarative content type. An imperative application is an application whose initial entity is of an imperative content type.

NCL defines the glue that holds media objects together in a multimedia presentation. NCL applications only define how media objects are structured and related, in time and space. As a glue language, NCL does not restrict or prescribe any media-object content type.

requirements. Based on scalable vector graphics (SVG) [20] and other extensions [19], LASeR has its focus on media synchronization, as well as NCL. Both languages support content and presentation adaptability, and provide support for live editing commands. NCL reuse facilities [21] are more versatile than LASeR's. NCL also offers more powerful support to applications targeting multiple exhibition devices. Despite being a good solution, mainly for mobile devices, LASeR does not have a commercial implementation yet.

NCL was the first standardized technology of the ITU-T multimedia application framework for IPTV services [3, 22]. Using NCL, an author can declaratively describe the spatial and temporal behavior of a multimedia presentation, associate hyperlinks (for viewer interactions) with media objects, define alternatives for content and content presentation (adaptation), and describe the presentation layout on multiple exhibition devices. In addition, NCL provides an API that allows building and modifying an application on the fly through live editing commands. NCL does not define any media itself. As a glue language, NCL bypasses legacy problems embedding other objects. Moreover, NCL supports a very efficient and lightweight scripting language when algorithmic computations are needed.

The Nested Context Language

NCL defines the glue that holds media objects together in a multimedia presentation. NCL applications only define how media objects are structured and related in time and space. As a glue language, NCL does not restrict or prescribe any media object content type. In this sense, media objects may be image objects (JPEG, PNG, etc.), video objects (MPEG, MOV, etc.), audio objects (MP3, WMA, etc.), text objects (TXT, PDF, etc.), imperative objects (Xlet, Lua, etc.), other declarative objects (HTML, LIME, SVG, nested NCL, etc.), and so on. Which media objects are supported depends on which media players are embedded in the NCL engine. It is worth mentioning that NCL treats main audiovisual streams like all other media objects it can relate. It should also be stressed that NCL treats an XHTML document as one of its possible media object types. Therefore, NCL does not substitute but embeds XHTML-based documents.

In the recently approved revision of ITU-R Recommendation BT.1699 [23], NCL was introduced as a glue language to harmonize declarative content for DTV. NCL can be viewed as a feasible solution to promote lightweight integration among most multimedia application technologies mentioned in the previous section.

NCL is an XML application that follows the modularization approach. Several NCL profiles have been defined. Of special interest is the Enhanced DTV (EDTV) Profile, defined for DTV.

Figure 1 shows an almost complete NCL element tree. Figure 3 exemplifies an NCL application, discussed below to clarify concepts and to illustrate the use of NCL elements.

The NCL Structure module defines the root element, `<ncl>`, and its child elements, `<head>` and `<body>`, following the terminology adopted by Word Wide Web Consortium (W3C) standards.

The `<head>` element may have `<importedDocumentBase>`, `<ruleBase>`, `<transitionBase>`, `<regionBase>`, `<descriptorBase>`, and `<connectorBase>` elements as its children. The `<body>` element

```
1:  <ncl>
2:    <head>
3:      <importedDocumentBase>
4:        <importNCL/>
5:      </importedDocumentBase>
6:      <ruleBase>
7:        <importBase/>
8:        <compositeRule/>
9:        <rule>
10:     </ruleBase>
11:     <transitionBase>
12:       <importBase/>
13:       <transition/>
14:     </transitionBase>
15:     <regionBase>
16:       <importBase/>
17:       <region>
18:         <region/>
19:       </region>
20:     </regionBase>
21:     <descreiptorBase>
22:       <importBase/>
23:       <descriptor>
24:         <descriptorParam/>
25:       </descriptor>
26:       <descriptorSwitch>
27:         <defaultDescriptor/>
28:         <bindRule/>
29:         <descriptor/>
30:       </descriptorSwitch>
31:     </descriptorBase>
32:     <connectorBase>
33:       <importBase/>
34:       <causalConnector>
35:         <connectorParam/>
36:         <compoundCondition/>
37:         <compoundAction/>
38:       </causalConnector>
39:     </connectorBase>
40:   </head>
41:   <body>
42:     <port/>
43:     <property/>
44:     <media>
45:       <area/>
46:       <property/>
47:     </media>
48:     <context>
49:       <port/>
50:       <property/>
51:       <media/>
52:       <context/>
53:       <link/>
54:       <switch/>
55:     </context>
56:     <switch>
57:       <defaultComponent/>
58:       <switchPort/>
59:       <bindRule/>
60:       <media/>
61:       <context/>
62:     <switch/>
63:     <link>
64:       <linkParam/>
65:       <bind>
66:         <bindParam/>
67:       </bind>
68:     </link>
69:   </body>
70: </ncl>
```

Figure 1. *Structure of NCL documents.*

may have <port>, <property>, <media>, <context>, <switch>, and <link> elements as its children. The <meta> and <metadata> elements, omitted in Fig. 1, may be child elements of <head>, <body>, and <context> elements. The <body> element is treated as a context object, as defined in what follows.

The <media> element defines a media object specifying its type and content location. Some special types are predefined by the language: application/x-ncl-settings type, specifying an object whose properties are global variables defined by the application author or reserved environment variables manipulated by an NCL user agent; application/x-ncl-time type, specifying a <media> element whose content is the Universal Time Coordinated (UTC); application/x-ncl-NCL type specifying a <media> element whose content is another NCL embedded application; and imperative media object types application/x-ncl-NCLua and application/x-ncl-NCLet, specifying <media> elements whose content is a code span written in Lua or Java languages, respectively.

The <context> element defines a context object. A context is a composite that contains a set of objects (media, context, or switch) and a set of links. Like the <body> element, <context> elements may have <port>, <property>, <media>, <context>, <switch>, and <link> child elements.

The <switch> element allows for the definition of alternative objects (represented by <media>, <context>, and <switch> elements) to be chosen in presentation time. Test rules used for choosing the switch component are defined by <rule> or <compositeRule> elements that are grouped by the <ruleBase> element, defined as a child of the <head> element.

NCL allows defining object interfaces that are used in relationships with other object interfaces. The <area> element allows the definition of a content anchor representing a spatial, temporal, or spatiotemporal segment in a media object's (<media> element) content, or a code span in imperative media objects. The <property> element is used for defining object properties (local variables) or a group of object properties as one of the object interfaces. The <port> element specifies a composite (<context>, <body>, or <switch> element) port together with its respective mapping to an interface of one of the composite child components. The <switchPort> element allows the creation of <switch> element interfaces that are mapped to a set of alternative interfaces of the switch's internal objects.

The <descriptor> element specifies temporal and spatial information needed to present each media object. The element may refer to a <region> element to define the initial position of a <media> element in some output device. The set of descriptors of a document is defined inside the <descriptorBase> element, a child of the <head> element. Also inside the <head>, the <regionBase> element defines a set of <region> elements for a given class of devices in a multidevice environment, as illustrated in Fig. 2 and discussed afterward. Each <region> may contain another set of nested <region> elements, and so on, recursively; regions define areas (e.g., screen windows), and are referred to by <descriptor> elements, as previously mentioned.

Figure 2. *NCL multidevice application.*

A <causalConnector> element represents a relation that may be used for creating <link> elements. In a causal relation, a condition role shall be satisfied in order to trigger an action role. Conditions and actions are defined as children of <link> elements. A <link> element binds (through its <bind> elements) an object's interfaces to connector roles (conditions or actions), defining a spatiotemporal relationship among objects (<media>, <context>, <body>, or <switch> elements).

The <descriptorSwitch> element contains a set of alternative descriptors to be associated with media objects. Analogous to the <switch> element, the <descriptorSwitch> choice is done during document presentation, using test rules defined by <rule> or <compositeRule> elements.

The <importBase> element allows any base to incorporate another already defined external base. Additionally, NCL documents may be imported as a whole through <importNCL> elements. The <importedDocumentBase> element specifies a set of imported NCL documents, and shall also be defined as a child element of the <head> element.

Some important NCL elements' attributes are defined in other NCL modules. The EntityReuse module allows an NCL element to be reused [21]. This module defines the refer attribute, which identifies the element that will be reused. Only <media>, <context>, <body>, and <switch> may be reused. The KeyNavigation module provides extensions necessary to describe focus movement operations (navigation control) among media objects. The Animation module provides the necessary extensions to describe what happens when a property value is changed. The change may be instantaneous, but it may also be carried out during an explicitly declared duration, either linearly or step by step. The Animation module defines attributes that may be incorporated by actions, defined as child elements of <causalConnector> elements.

The NCL <transitionBase> element specifies a set of transition effects, defined by

<transition> elements, and shall be defined as a child element of the <head> element. Transition effects can be applied at the beginning or end of a media object presentation, as defined in attributes of the <descriptor> element associated with the media object.

In order to illustrate an NCL use case, assume the following simple application: during a soccer game animation, an advertisement, temporally related with a special scene of the animation, is presented, allowing a viewer to interact to buy the product (soccer shoes). In order to avoid annoying other viewers, the interaction processes will not be exhibited on the TV screen, but on secondary device screens, for example, mobile phones. Figure 2 illustrates this application. The complete NCL code is presented in Fig. 3.

In Fig. 3, lines 5 to 9 define two regions in the base device (the TV set). The "mainScreenRg" region occupies the whole display and the "iconRg" region (with zIndex="1") overlays the previous region (default zIndex="0") in the bottom right corner of the device screen. Lines 10 to 12 define the "advertRg" region as filling the whole "systemScreen(2)" secondary display. Lines 13 to 18 define the descriptor base, whose <descriptor> elements define in which region each related media object will be exhibited. The "iconDs" descriptor also specifies the explicit duration of 40 s for the icon exhibition.

```
1:  <?xml version="1.0" encoding="ISO-8859-1"?>
2:  <ncl id="merchandisingDocument"
3:    xmlns="http://www.ncl.org.br/NCL3.0/EDTVProfile:>
4:  <head>
5:    <regionBase>
6:      <region id="mainScreenRg"/>
7:      <region id="iconRg" bottom="10%" right="10%"
8:                width="20% height="20%" zIndex="1"/>
9:    </regionBase>
10:   <regionBase device="systemScreen(2)">
11:     <region id="advertRg"/>
12:   </regionBase>
13:   <descriptorBase>
14:     <descriptor id="mainScreenDs" region="mainScreenRg"/>
15:     <descriptor id="iconDs" region="iconRg"
16:                 explicitDur="40s"/>
17:     <descriptor id="advertDs" region="advertDs"/>
18:   </descriptorBase>
19:   <connectorBase>
20:     <causalConnector id'"onBeginStart">
21:      <simpleCondition role="onBegin"/>
22:      <simpleAction role="start"/>
23:     </causalConnector>
24:     <causalConnector id="onKeySelectionStart">
25:       <connectorParam name="keyCode"/>
26:       <simpleCondition role="onSelection"key="$keyCode"/>
27:       <simpleAction role="start"/>
28:     </causalConnector>
29:   </connectorBase>
30: </head>
31: <body>
32:   <port id="mainP" component="video"/>
33:   <media id="video" descriptor-"mainScreenDs"
34:            src="rtp://www.ginga.org.br/video.mp4">
35:       <area id="iconA" begin="10s" end="50s"/>
36:   </media>
37:   <media id="icon" descriptor="iconDs"
38:            src=" blinkAnimation.lua"/>
39:   <media id="advert" descriptor="advertDs"
40:            src="http://www.ginga.org.br/buy.xhtml"/>
41:   <link xconnector="onBeginStart">
42:     <bind role="onBegin" component="video"
43:            interface="iconA"/>
44:     <bind role="start" component="icon"/>
45:   </link>
46:   <link xconnector="onKeySelectionStart">
47:     <linkParam name="keyCode" value="RED"/>
48:     <bind role="onSelection" component="icon"/>
49:     <bind role="start" component="advert"/>
50:   </link>
51: </body>
52: </ncl>
```

Figure 3. *NCL application code.*

Lines 19 to 29 define the connector base. The causal connector "onBeginStart" has its condition satisfied when a media object's anchor starts its presentation, triggering, as a result, the starting presentation action on another media object's anchor. The causal connector "onKeySelectionStart" specifies that the selection of a media object's anchor, by pressing a key, triggers the starting presentation action on another media object's anchor.

The <body> element defines the document structure. The port in line 32 states from which media object the document presentation initiates (in this case, the "video" media object). The "video" media object (lines 33 to 36) is received from "rtp://www.ginga.org.br/video.mp4" to be presented in full screen, as defined by its descriptor. The "icon" is an imperative media-object with Lua code, implementing a blinking soccer shoes image. The "advert" media object (lines 39 and 40) is an XHTML content to be presented on secondary devices.

Lines 41 to 45 define the first relationship, stating that the "icon" media object must start its exhibition as soon as the "iconA" temporal anchor of the "video" media object begins its presentation. Lines 46 to 50 specify another relationship, establishing that a viewer interaction with the "icon" image pressing the remote control RED key must start "advert" media object presentation (a purchase web page) on secondary devices.

The Lua code that carries out the soccer shoes animation is presented in Fig. 4.

The isON and stop flags (line 1) control when the soccer shoes icon is presented and when the animation stops, respectively. The function myBlinkAnimation is used to redraw the canvas, performing the blink effect on the icon. The function starts getting the canvas size (line 3), draws a rectangle filling the canvas (line 4), and then combines the soccer shoes image with the canvas, depending on the isON flag value (lines 5 to 7). Line 8 updates the isON flag, and line 9 calls the canvas flush, in order to update the screen. Line 10 calls myBlinkAnimation every 1000 ms, while stop flag is true.

The NCLua API defines an event-oriented communication between NCL and Lua codes. When the Lua object is started, line 18 registers an event handler for the stop presentation event to be received from NCL, and starts the blinking animation (line 19), calling the myBlinkAnimation function. When an NCL link stops the Lua object, a "stop" event is received and treated by lines 13 to 16.

THE GINGA-NCL ARCHITECTURE

Ginga-NCL was initially proposed for terrestrial

DTV systems. Then the same architecture and facilities were applied to IPTV. The modular architecture of Ginga-NCL also allows for its use with other transport systems (e.g., satellite and cable TV). Figure 5 depicts the Ginga-NCL components and how they relate with other components of a general IPTV architecture.

The Ginga-NCL Presentation Engine is a logical subsystem responsible for running NCL applications. The core of the Presentation Engine is the Formatter. This component is in charge of receiving and controlling multimedia applications written in NCL. Applications are delivered to the Formatter by the Ginga Common Core (Ginga-CC) subsystem.

Upon receiving an application, the Formatter requests the XML Parser and Converter components to translate the NCL specification to the Ginga-NCL internal data structures necessary for controlling the application presentation. From then on, the Scheduler component is started in order to orchestrate the presentation. The prefetching of a media object's content, the evaluation of link conditions, and the scheduling of corresponding link actions that guide the presentation flow are tasks performed by the Scheduler. In addition, the Scheduler is responsible for commanding the Player Manager component to instantiate appropriate players, according to the media content types to be exhibited. Media contents are acquired through the protocol stack, and they can come from different communication networks.

A generic API establishes the necessary communication between players and the Presentation Engine (Scheduler component). Thanks to this API, Ginga-NCL and Ginga-CC are strongly coupled but independent subsystems. Ginga-CC can be substituted by other third party implementations for IPTV engines, allowing Ginga-NCL to be integrated with other IPTV middleware, extending their functionalities with facilities to support NCL DTV applications.

Players are responsible for notifying the Presentation Engine of events defined in NCL applications, that is, when a media segment (an anchor) begins or ends its presentation, or when it is selected. Players that do not follow the generic API must use the services provided by Adapters.

In Ginga-NCL a DTV application can be generated or modified on the fly using NCL editing commands. The Presentation Engine deals with NCL applications collected inside a data structure known as a private base. A Private Base Manager component is in charge of receiving NCL editing commands and maintaining the NCL documents being presented. NCL editing commands [3] are divided into three subsets. The first one focuses on private base activations and deactivations (`openBase`, `activateBase`, `deactivateBase`, `saveBase`, and `closeBase` commands). In a private base NCL applications can be started, paused, resumed, stopped, and removed, through well defined editing commands that compose the second subset. The third subset defines commands for updating an application on the fly, allowing NCL elements to be added and removed, and allowing values to be assigned to NCL `<property>` elements.

```
1:  local isON, stop = true, false
2:  function myBlinkAnimation()
3:    local w,h = canvas:attrSize()
4:    canvas:drawRect('fill', 0, 0, w, h)
5:    if isON then
6:       canvas:compose(0, 0, canvas:new("soccer_shoes.png"))
7:    end
8:    isON = not isON
9:    canvas:fluch()
10:   if not stop then event.timeer(1000, myBlinkAnimation) end
11: end
12: function handler(evt)
13:   if evt.class == 'ncl' and evt.type == 'presentation'
14:                                    and evt.action == 'stop' then
15:         stop=true
16:   end
17: end
18: event.register(handler)
19: myBlinkAnimation()
```

Figure 4. *Lua application code.*

The Ginga-NCL Presentation Engine supports multiple presentation devices through its Layout Manager component. This component is responsible for mapping all regions defined in an NCL application to regions on exhibition devices.

The Context Manager component of Ginga-CC is responsible for gathering platform characteristics and viewer profiles into a database used to update an NCL application's global variables defined in the NCL settings object, presented in the previous section. This information can then be used to adapt an application.

The display graphical model defined by the receiver platform is maintained by the Graphics Manager component, which is in charge of handling operations on graphic planes and zIndex overlay requests.

The Data Processing component offers support for acquiring data transported in DSM-CC carousels [10] or other pushed data protocols. The Persistency component is in charge of every data storage management requested by applications. The Tuner component is responsible for offering an API for RF or IPTV channel management (broadcast RF frequency tuning or multicast group entry, respectively). The Search Engine component can be activated by an electronic program guide (EPG) or another service that needs data mining. Security management is performed by the DRM and Conditional Access (CA) components. Finally, each component of Ginga can be updated through the Update Manager component.

IPTV-specific services, such as VOD and datacasting, are outside the Ginga-NCL scope. However, an API is defined offering NCL services to these IPTV service components. Thus, a VoD service can, for example, play an NCL DTV application besides the main audiovisual stream requested. In addition, a whole VoD service can be written in NCL/Lua, as well as other IPTV services like widget portals, gaming, and EPGs.

Final Remarks

The Ginga-NCL open source reference implemen-

There are several commercial implementations of Ginga-NCL for terrestrial set-top boxes. Some of them also offer support to IPTV platforms. Some commercial set-top boxes plan to offer Ginga-NCL support both for IPTV and for satellite TV.

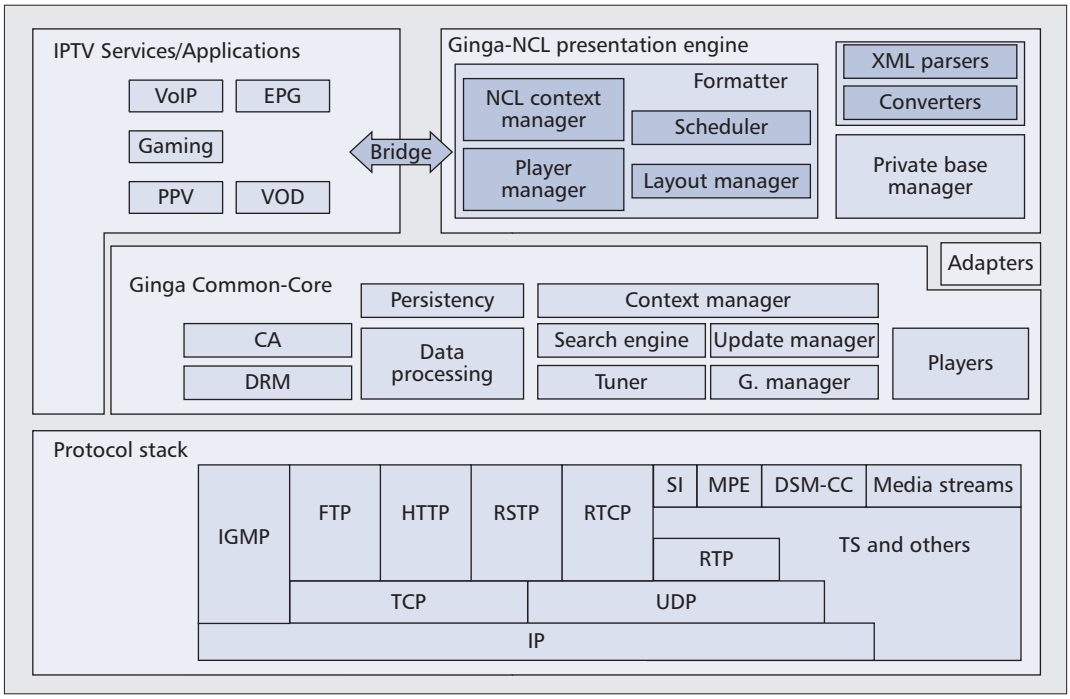

Figure 5. *The Ginga-NCL architecture for IPTV platforms.*

tation can be obtained from http://www.ncl.org.br.

There are several commercial implementations of Ginga-NCL for terrestrial set-top boxes. Some of them also offer support for IPTV platforms. Some commercial set-top boxes plan to offer Ginga-NCL support for both IPTV and satellite TV.

In agreement with the ITU-T multimedia application framework for IPTV [22], this article proposes Ginga-NCL integration with third party IPTV middlewares, extending their API to support NCL DTV applications. The integration can be done through adapting Ginga-CC to IPTV platforms or adapting the third-party IPTV middleware's core to provide the API requested by Ginga-NCL.

Several advantages come from Ginga-NCL integration to IPTV platforms. First, Ginga-NCL provides a powerful declarative language targeted to the DTV application domain, unlike all other DTV declarative middleware specifications, which are based on general-purpose languages or web technologies.

Second, NCL applications are easier to design and usually do not require programming expertise, as imperative language approaches do. Imperative approaches occasionally put application portability at risk, presentation control is much more difficult to achieve as a rule, and they are more prone to errors committed by application programmers.

Third, an expressive declarative language such as NCL can support almost all usual DTV applications, and for those that do not match the NCL model focus, NCL supports the efficient and lightweight Lua scripting language.

Finally, it is possible to build hybrid receivers supporting both terrestrial DTV and IPTV (and other DTV systems as well), decreasing receiver costs and offering both services to users. The glue-language approach of NCL is an efficient and suitable solution in such a hybrid scenario where harmonization is desirable and now demanded.

References

[1] ABNT NBR, "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"; http://www.abnt.org.br/imagens/Normalizacao_TV_Digital/ABNTNBR15606-2_2007Ing_2008.pdf

[2] R. Ierusalimschy, *Programming in Lua*, 2nd ed., Lua.org, 2006.

[3] ITU-T Rec. H.761, "Nested Context Language (NCL) and Ginga-NCL for IPTV Services," Geneva, Apr. 2009.

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

[5] ETSI Std. TS 102 812, "Digital Video Broadcasting (DVB), Multimedia Home Platform (MHP) Specification," v. 1.1.1, 2003.

[6] ATSC Std., "Advanced Application Platform (ACAP)," Doc. A/101, 2005.

[7] ARIB STD-B24 v. 3.2, "Volume 3: Data Coding and Transmission Specification for Digital Broadcasting," 2002.

[8] "IPTV Middleware Market Dynamics," *Light Reading Insider*, vol. 6, no 9.

[9] Microsoft, "Microsoft Mediaroom"; http://www.microsoft.com/mediaroom/

[10] Minerva Networks; http://www.minervanetworks.com

[11] Orca Interactive; http://www.orcainteractive.com

[12] Myrio and Nokia Siemens Networks; http://www.myrio.com

[13] Soft At Home; http://www.softathome.com

[14] Jornada en la Cátedra Alcatel-Lucent, "IPTV Trends," Madrid, Spain 2009.

[15] ITU-T Rec. H.762, "Lightweight Interactive Multimedia Environment," Geneva, Dec. 2009.

[16] HbbTV, "HbbTV Overview," EBU/ETSI Hybrid Broadcast Broadband Wksp., Amsterdam, 2009.

[17] The Project Canvas Wiki; http://www.projectcanvas.co.uk

[18] Verizon FiOS TV Development Resources; https://www22.verizon.com/fiosdeveloper/General/Resource.aspx

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

[20] W3C Rec., "Scalable Vector Graphics — SVG 1.1 Specification," 2003; http://www/w3/org/TR/SVG11

[21] L. F. G. Soares and C. S. Soares Neto, "Nested Context Language 3.0 — Reúso e Importação," tech. rep., Informatics Dept., no. 33, 2009.
[22] ITU-T Rec. H.760, "Overview of Multimedia Application Frameworks for IPTV," Geneva, Apr. 2009.
[23] ITU-R Rec. BT-1699, "Harmonization of Declarative Content Format for Interactive TV Applications," Geneva, 2009.

Additional Reading

[1] ISO/IEC Std. 13818-6, "Information Technology — Generic Coding of Moving Pictures and Associated Audio Information — Part 6: Extensions for DSM-CC," 1998.

Biographies

Luiz Fernando Gomes Soares (lfgs@inf.puc-rio.br) is a full professor in the Informatics Department of the Pontifical Catholic University of Rio de Janeiro (PUC-Rio), Brazil, where since 1990 he has headed the TeleMídia Lab. He is a board member of the Brazilian Internet Steering Committee and Chair of the Middleware Working Group for the Brazilian Digital TV System.

Marcio Ferreira Moreno (mfmoreno@inf.puc-rio.br) received his M.Sc. degree from the Informatics Department of PUC-Rio, Brazil, in April 2006. At present, he is a Ph.D. student at PUC-Rio and an associate researcher in the TeleMídia Lab. He has worked on the terrestrial Brazilian DTV specifications and the Ginga-NCL reference implementation.

Carlos de Salles Soares Neto (csalles@inf.puc-rio.br) is an assistant professor at the Federal University of Maranhão (UFMA). He received his M.Sc. degree from the Informatics Department of PUC-Rio in August 2003. At present, he is a Ph.D. student at PUC-Rio and an associate researcher in the TeleMídia Lab. He has worked on the terrestrial Brazilian DTV specifications.

Marcelo Ferreira Moreno (moreno@inf.puc-rio.br) is an