Ginga-NCL: Relating Imperative, Declarative and Media Objects

Marcio Ferreira Moreno
Departamento de Informática – PUC-Rio
Rua Marquês de São Vicente, 255
Rio de Janeiro – 22453-900 – Brazil
mfmoreno@inf.puc-rio.br

ABSTRACT
The process of developing a declarative middleware for interactive digital TV systems presents significant challenges. The main goal of this PhD thesis is to propose and to develop an NCL presentation environment for interactive digital TV systems. As one of its by-product, a declarative middleware named Ginga-NCL was developed, becoming the reference implementation of the Brazilian Terrestrial Digital TV System. This paper focuses on how NCL relationships among "distributed" applications are guaranteed by Ginga-NCL.

Categories and Subject Descriptors
I.7.2 [**Document Preparation**]: Languages and systems, markup languages, multimedia, hypermedia, standards. D.3.2 [**Language Classifications**]: Specialized application languages

General Terms
Design, Standardization, Languages.

Keywords
Ginga-NCL, NCL, DTV, Middleware, Synchronism, SBTVD.

1. INTRODUCTION
Interactive Digital TV (iDTV) applications can be understood as hypermedia documents in which related media objects of different types compose the scenes presented in receiver devices. NCL (Nested Context Language) [1], an XML based language, has become the Brazilian solution for its iDTV standard not only due to its power for easily defining spatial and temporal relationships among media objects, including viewer interactions, but also due to its facility for defining structured distributed applications across multiple exhibition devices, its support to content and presentation adaptations, its provisioning for live iDTV application producing, and its support to content and structure reuse.

The main goal of this PhD thesis is to propose and to develop an NCL presentation environment for iDTV systems. A declarative middleware named Ginga-NCL was developed as one of its results, becoming the reference implementation of the Brazilian Terrestrial Digital TV System[1]. The Ginga-NCL integration in IPTV and DTV systems has brought some new solutions to several issues regarding the support to iDTV applications, as discussed in Section 3.

An NCL document (a Ginga-NCL application) only defines 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, we can have the usual media objects (text, image, video, audio, etc), imperative objects (objects with imperative code content) and declarative objects (objects with declarative code content), as NCL media objects. The media objects supported depends on the media players that are integrated in the NCL presentation engine.

Although Ginga-NCL reference implementation includes SMIL (Synchronized Multimedia Integration Language) support, the Brazilian Terrestrial Digital TV System (SBTVD) requires only the support to XHTML and NCL declarative objects. As for imperative media objects, SBTVD requires support to Lua [2] and Java (Xlet) [3] in its reference implementation. As a consequence, it is possible to create NCL applications that can use other applications (as NCL media objects) specified using the aforementioned declarative and imperative languages as well as to specify synchronization relationships among these applications.

The current research of this PhD thesis focus on both imperative and declarative object types used in NCL applications: how they can be defined, how they can be related, and how to specify the expected behavior of imperative and declarative engines through the Ginga-NCL architecture.

Considering the NCL support for multiple devices, distributed application processing allows multi-viewer interactions coming from different devices, which will allow several new interaction experiences. Another work in progress is how relationships among these "distributed" applications will be guaranteed. The multiple device support complements this thesis directions and it has been done in association with another work in progress [4].

The remainder of this paper is organized as follows. Section 2 discusses some related work. Section 3 presents the Ginga-NCL architecture and the thesis's current contributions. Section 4 discusses future directions concerning imperative and declarative objects. Section 5 is reserved for final remarks.

2. RELATED WORK
The iDTV applications can be partitioned in a set of declarative applications and a set of imperative applications [3]. A declarative application is an application whose initial entity is of

[1] An open source reference implementation of Ginga-NCL is available under the GPLv2 license: www.gingancl.org.br/index_en.html

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 application paradigms. Generally, the imperative environment of these systems is based on the use of a Java virtual machine and the definition of generic APIs that provide access to the iDTV receiver's typical resources and facilities. Usually, the declarative environment is based on XHTML user agents with ECMAScript and DOM support [3]. XHTML content authors can embed Xlets within their XHTML documents, or access Java constructs from an ECMAScript-to-Java bridge.

XHTML is a media-based declarative language, which means that the structure defined by the relationships among XHTML objects (XHTML documents or objects embedded in XHTML documents) is inserted in the document's media content. Reference relationships defined by XHTML links are the focus of the XHTML declarative language. Other relationship types, like spatio-temporal synchronization relationships and alternative relationships (media adaptations), are usually defined using ECMAScript; thus they cannot take profit of the easy authoring and less error prone way offered in other declarative languages, like NCL and SMIL [5].

Unlike XHTML, NCL and SMIL have a stricter separation between content and structure, and they provide a non-invasive control of presentation linking and layout. SMIL allows the inclusion of media objects into a SMIL document, although it does not define the use of imperative and declarative objects, neither provides support for live editing and distributed processing.

3. WORK ALREADY ACCOMPLISHED

As in all main terrestrial DTV Systems [3], the Brazilian middleware, named Ginga, supports both declarative applications (through its presentation, or declarative, environment named Ginga-NCL) and imperative applications (through its execution, or imperative, environment named Ginga-J [6]). Figure 1 depicts the modular architecture defined in Ginga-NCL reference implementation of SBTVD and how each module relates with other components of DTV receivers.

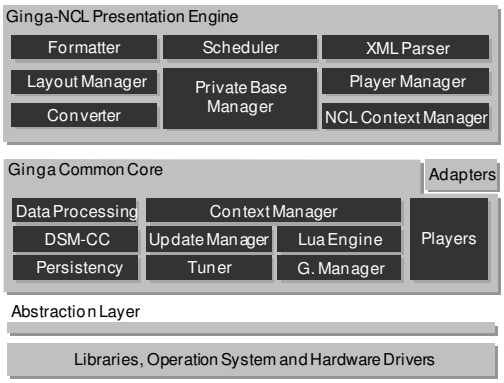

Figure 1. Ginga-NCL Architecture

Ginga-NCL Presentation Engine is a logical subsystem responsible for running NCL applications. In order to provide common digital TV services for both Ginga-NCL and Ginga-J, the Ginga Common Core subsystem was defined.

The core of Ginga-NCL 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 subsystem. Upon receiving an application, the Formatter requests the XML Parser and Converter components to translate the NCL application 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 NCL document presentation. The pre-fetching of media object's contents, the evaluation of link conditions and the scheduling of corresponding relationships actions that guide the presentation flow are some tasks performed by the Scheduler component. In addition, the Scheduler component is responsible for command the Player Manager component to instantiate an appropriate Player, according to the media content type to be exhibited in a given moment in time. Media contents are acquired through specifics protocol stacks, and can come from different communication networks. The author-specified relationships among media objects are respected during NCL presentations, no matter if their contents are pushed by broadcasting or pulled on demand.

Ginga-NCL was developed in a way that it can easily integrate a variety of object types. A generic API was defined to establish the necessary communication between Players components and the Presentation Engine (Scheduler component). Thanks to this API, the Ginga-NCL Presentation Engine and the Ginga core are strongly coupled but independent subsystems.

Players are responsible for notifying the Presentation Engine about object events defined in NCL applications, that is, when a media segment (an anchor) begins and ends its presentation, when an anchor is selected, or when an object property changes its value. Presentation events can be derived from content timestamps (for example, PTS in MPEG-2 [7]), timers started with static contents, etc., depending on the media format. An important contribution of Ginga-NCL implementation is how to handle timebases of media contents multiplexed in an elementary stream. Algorithms to process multiple timebases multiplexed in DSM-CC NPT elementary stream [8] were defined, allowing relating interlaced stream contents with other NCL media objects.

Players that do not follow the specified Ginga API must use services provided by Adapters. Any user agent or execution engine could be adapted to the Ginga-NCL Players, e.g. XHTML browsers, SMIL players, Java TV engine, etc.

Another major contribution of this thesis [9] is the introduction of Ginga-NCL support for live editing applications. A DTV application can be generated or modified on the fly, using Ginga-NCL editing commands [9]. These commands can be sent (or retrieved) by the service provider through a protocol stack or can be issued by imperative objects embedded in an NCL application.

The Presentation Engine deals with NCL applications collected inside a data structure known as private base. A Private Base Manager component is in charge of receiving NCL document editing commands and maintaining the NCL documents being presented. The set of NCL live editing commands [9] are divided in three subsets.

The first one focuses on the private base activation and deactivation (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 set to media object properties. Ginga-NCL editing commands are defined using an XML-based syntax notation identical to that used by the NCL document [1].

Ginga-NCL Presentation Engine supports multiple presentation devices through its Layout Manager module. This component is responsible for mapping all regions defined in an NCL application to canvas on receiver's exhibition devices. Currently, Resende Costa [4] thesis aims at defining a set of data structures focusing in multiple device use cases (for exhibition and also for distributed processing).

Moving back the attention to the Ginga Common Core, the Context Manager component is responsible for gathering platform characteristics and viewer profile into a data base used to update the NCL application's global variables defined in the NCL special media object called settings node. This information can then be used to adapt an application content or presentation. The settings node can also be used as any other NCL media object as an actor of a relationship.

The Players component of Ginga Common Core are composed by content decoder/players and procedures to obtain contents transported in any network supported by a specific protocol stack. 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 (five in the case of SBTVD and ISDB), including overlay requests.

In the case of terrestrial DTV, the DSM-CC and Data Processing components offer support for acquiring data transported in DSM-CC carousels [8]. A resource identification mechanism was created in order to relieve application authors from having knowledge of the addressing scheme used to identify resources in DTV presentation environments and transport systems, which is also another important contribution of this thesis [10]. The Persistency component is in charge of every data storage management requested by applications. The Tuner component is responsible to offer an API for TV channels management. Finally, an Abstraction Layer was defined in order to hide implementation details, such as graphical device model, hardware decoder drivers, etc, to the Ginga-NCL and Ginga Common Core implementations

Each component of Ginga-NCL can be updated through the Update Manager component. Another contribution of this thesis is the component based implementation of Ginga-NCL, allowing component update and component loading during runtime.

4. FUTURE DIRECTIONS

As aforementioned, Ginga-NCL supports XHTML, SMIL and also nested NCL documents as NCL declarative objects. Ginga-NCL also supports Lua and Java Xlet as NCL imperative objects. NCL functionalities allow a simple definition of relationships among these objects such as, for example, "run a Lua function when a specific video segment is shown" or "submit an XHTML form when an Xlet class gets instantiated". Figure 2 depicts a more detailed example. This NCL application is used in this section in order to clarify concepts and to illustrate the use and the specification of relationships among declarative, imperative and other media objects, which are the current focus of this thesis.

```
<?xml version="1.0" encoding="UTF-8"?>
<ncl id="SynchronizingCodes"
xmlns="http://www.ncl.org.br/NCL3.0/EDTVProfile">

<head>
  <regionBase device="default">
   <region id="hReg" width="20%" height="20%"
right="5%" bottom="5%"/>
  </regionBase>

  <regionBase device="systemScreen(2)">
   <region id="dReg" width="80%" height="80%"
top="10%" left="10%"/>
  </regionBase>

  <descriptorBase>
   <descriptor id="hDesc" region="hReg"/>
   <descriptor id="pDesc" region="dReg"/>
   <descriptor id="gDesc" region="dReg"/>
  </descriptorBase>

  <connectorBase>
   <causalConnector id="onBeginStart">
    <simpleCondition role="onBegin"/>
    <simpleAction role="start"/>
   </causalConnector>

   <causalConnector id="onEndStart">
    <simpleCondition role="onEnd"/>
    <simpleAction role="start"/>
   </causalConnector>
  </connectorBase>
</head>

<body>
 <port id="pHandler" component="handler"/>
 <port id="pGame" interface="phase1"
component="game"/>

 <media id="handler" src="scripts/handler.lua"
descriptor="hDesc">
  <property id="inc"/>
  <area id="createNextLevels"/>
 </media>

 <media id="promo" src="advert/promo.xhtml"
descriptor="pDesc">
  <area id="buyNextLevels"/>
 </media>

 <media id="game" src="game/game.ncl"
descriptor="gDesc">
  <area id="level1"/>
 </media>

 <link id="l1" xconnector="onBeginStart">
  <bind component="promo" interface="buyNextLevels"
role="onBegin"/>
  <bind component="handler"
interface="createNextLevels" role="start"/>
 </link>

 <link id="l1" xconnector="onEndStart">
  <bind component="game" interface="level1"
role="onEnd"/>
  <bind component="promo" role="start"/>
 </link>
</body>
</ncl>
```

Figure 2. Example of an NCL Application

Figure 2 presents the NCL structure module defining the root element, called <ncl>, and its child elements, the <head> and the <body> elements, following the terminology adopted by other W3C standards. The <head> element defines the following child

elements: <regionBase>, <descriptorBase> and <connectorBase>. There are two <regionBase> elements. Each one is associated with a particular exhibition device class[2] where presentation will take place. The <descriptorBase> element contains <descriptor> elements, each one referring a <region> element in order to define the initial exhibition area of a <media> element. A <causalConnector> element is specified as child of <connectorBase> element in order to define a causal relation that may be used to create causal relationships defined by <link> elements. In a causal relation, a condition shall be satisfied in order to trigger an action. Conditions and actions are specified using <role> child elements of <causalConnector> elements [1].

The <body> element includes <port>, <media>, and <link> child elements. A <port> element state from which media objects a document presentation chain must initiate (in the example, the "handler" and "game" objects). In Figure 2, <media> elements specify imperative (a Lua object) and declarative (an XHTML and an embedded NCL object) objects and its content location. Finally, the <link> element binds (through its <bind> elements) nodes interfaces with connector roles, defining a spatial and temporal relationship among objects.

The application starts with the presentation of a nested NCL object and a Lua object. The Lua object is presented on the receiver's default screen (usually the TV set). The NCL object is a game and is presented on a class of exhibition devices named systemScreen(2). All devices registered in this class shall run the game. There are two types of device classes: those able to run object players (active classes), and passive classes, which are only able to exhibit content processed in other devices. In SBTVD, class (1) is predefined as passive and class (2) as active.

The NCL game was designed with just one level of difficulty. If viewers playing the game in systemScreen(2) devices finish the first level, an XHTML object is started, also in systemScreen(2) devices, offering viewers a form of buying next game's stages. If a viewer buys the next game's stages, submitting the XHTML form, an XHTML embedded ECMAScript code starts the "buyNextLevels" interface. As a consequence, the Lua object interface, named "createNextLevels", is started. This Lua object interface is mapped to a Lua function that uses the editing command API of Ginga-NCL. Thus, editing the NCL game on-the-fly, the Lua code creates and enables the next stages of the NCL game.

It is important to point out that the imperative, or declarative, player is responsible for defining how the interfaces of its NCL object type are integrated to an NCL object code chunk. In the example, SBTVD [1] specifies how the Lua player should behave to mapping the "createNextLevels" interface into a "handler.lua" code chunk (the Lua function that uses the editing command API). Another interesting work in progress is to standardize how any imperative, or declarative, player should define this code-to-interface mapping.

[2] In NCL, exhibition devices can register themselves in specifics classes of a domain [1] [4].

The discussed NCL application can exemplify the challenges in guarantee the relationships among NCL media objects.

5. FINAL REMARKS

The open source reference implementation of Ginga-NCL is the main contribution of this PhD thesis. During Ginga-NCL design and implementation several issues were addressed as discussed in this paper. Although initially designed for terrestrial DTV systems, the modular implementation of Ginga-NCL allows its adaptation to other platforms. Some work is currently being done in this direction, both for IPTV and peer-to-peer TV.

6. ACKNOWLEDGMENTS

I would like to thank my advisor Professor Luiz Fernando Gomes Soares for its guidance. This work is being supported by CNPq, TeleMidia Lab and PUC-Rio.

7. REFERENCES

[1] 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. Available: http://www.abnt.org.br/imagens/Normalizacao_TV_Digital/ABNTNBR15606-2_2007Ing_2008.pdf

[2] Ierusalimschy, R. Programming in Lua. Lua.org, 2008.

[3] Morris, S., Smith-Chaigneau, A. "Interactive TV Standards", 2005, Elsevier Inc.

[4] Resende Costa, R. M. 2009, Synchronization Management for DTV Applications. Submitted for publication in Proceedings of the 2009 EuroITV Doctoral Consortium.

[5] "Synchronized Multimedia Integration Language (SMIL 3.0)", W3C Recommendation 01 December 2008. Avaiable at: http://www.w3.org/TR/SMIL3/

[6] Digital Terrestrial Television - Data Coding and Transmission Specification for Digital Broadcasting - Part 4: Volume 4 – GINGA-J: Environment for the execution of procedural applications.

[7] ISO/IEC 13818-1. Information technology – Generic coding of moving pictures and associated audio information - Part 1: Systems. ISO Standard, 2007.

[8] ISO/IEC 13818-6. Information technology – Generic coding of moving pictures and associated audio information - Part 6: Extensions for DSM-CC. ISO Standard, 1998.

[9] Resende Costa, R. M., Moreno, M. F., Rodrigues, R. F., and Soares, L. F. 2006. Live editing of hypermedia documents. In Proceedings of the 2006 ACM Symposium on Document Engineering (Amsterdam, The Netherlands, October 10 - 13, 2006). DocEng '06. ACM, New York, NY, 165-172. DOI= http://doi.acm.org/10.1145/1166160.1166202.

[10] Moreno, M. F., Rodrigues, R. F., and Soares, L. F. 2007. A Resource Identification Mechanism for Interactive DTV Systems. In Proceedings of the Ninth IEEE international Symposium on Multimedia Workshops (December 10 - 12, 2007). ISMW. IEEE Computer Society, Washington, DC, 215-220. DOI= http://dx.doi.org/10.1109/ISMW.2007.6.