Ginga-NCL: the Declarative Environment of the Brazilian Digital TV System

Luiz Fernando Gomes Soares, Rogério Ferreira Rodrigues, Márcio Ferreira Moreno

Department of Informatics
Catholic University of Rio de Janeiro
Rua Marquês de São Vicente 225
Phone: +55 (21) 3527-1530 (FAX)
Zip 22453-900 – Rio de Janeiro - RJ - BRAZIL
{lfgs—rogerio—marcio}@telemidia.puc-rio.br

Abstract

As in all main terrestrial DTV Systems, the Brazilian middleware, named Ginga, supports both declarative applications (through its presentation, or declarative, environment Ginga-NCL) and procedural applications (through its execution, or procedural, environment Ginga-J). Since hybrid applications are common, either type of Ginga application may make use of facilities of both presentation and execution application environments. This paper focuses on the presentation environment Ginga-NCL. The main Brazilian inovations are then presented, regarding the Ginga architecture, the declarative NCL language specification, the editing commands for live application production, and the transport data structure.

Keywords: Ginga, NCL, middleware, declarative environment, digital TV

1. INTRODUCTION

The universe of DTV (Digital TV) applications may be partitioned into a set of declarative applications and a set of procedural applications. A declarative application is an application whose initial entity is of a declarative content type. A procedural application is an application whose initial entity is of a procedural content type. A declarative content type is based on a declarative language, that is, a language that emphasizes the declarative description of a problem, rather than the decomposition of the problem into an algorithm implementation, as is done in the procedural programming style.

A purely declarative application is one whose every entity is of a declarative content type. A purely procedural application is one whose every entity is of a procedural content type. A DTV application is said to be hybrid when its entity set contains entities of both declarative and procedural content types.

In general, applications need not be purely declarative or procedural. In particular, declarative applications often make use of script content, which is procedural in nature. Furthermore, a declarative application may reference an embedded procedural code (for example, in a DTV domain, a Java TV Xlet). Similarly, a procedural application may reference declarative content, or may construct and initiate the presentation of declarative content.

Declarative languages are usually designed for a specific domain, aiming at a specific focus. When the

application focus matches the declarative language focus, the declarative paradigm is favored, since it provides a higher abstraction level, closer to the specification than procedural implementations. However, usually the declarative language focus is very restricted, leaving space for procedural language use. This is why DTV Systems usually provides support for both programming paradigms.

Thus, as in all main terrestrial DTV Systems, the Brazilian middleware, named Ginga, supports both declarative applications (through its presentation, or declarative, environment named Ginga-NCL) and procedural applications (through its execution, or procedural, environment named Ginga-J). Since hybrid applications are common, either type of Ginga application may make use of facilities of both presentation and execution application environments.

This paper focuses on the presentation environment Ginga-NCL. Section 2 briefly describes the Ginga architecture, discussing how Ginga-NCL relates with the other two major modules that compose the middleware Ginga: the Ginga Common-Core and the Ginga-J environment. Section 3 begins historically presenting some related work, showing their limitations regarding support to declarative applications. Then the declarative NCL language (the basis of Ginga-NCL) characteristics are presented. Section 4 gives an overview of the Ginga-NCL editing commands used to load and control an NCL document presentation. Section 5 introduces the Ginga Common-Core module, discussing the support it provides for Ginga-NCL to handle NCL documents' data structure. This module is responsible for receiving and treating application data from the Transport System, which is also discussed in Section 5. Section 6 brings a brief outline of the NCL scripting language, the Lua Language, calling attention to its Application Programming Interfaces (APIs). Section 7 exposes how Ginga-NCL and Ginga-J can relate with each other in order to support hybrid Java-NCL applications. Finally, Section 8 presents the paper concluding remarks.

2. GINGA ARCHITECTURE

The Ginga Architecture can be divided in three major modules: Common Core, Ginga-NCL and Ginga-J, the latter two composing the Ginga Specific Service, as shown in Figure 1.

Ginga-NCL is a logical subsystem of the Ginga System that processes NCL documents. A key component of Ginga-NCL is the declarative content decoding engine (NCL formatter named Maestro). Other important modules are the XHTML-based user agent, which includes a stylesheet (CSS) and ECMAScript interpreters, and the Lua engine, which is responsible for interpreting Lua scripts.

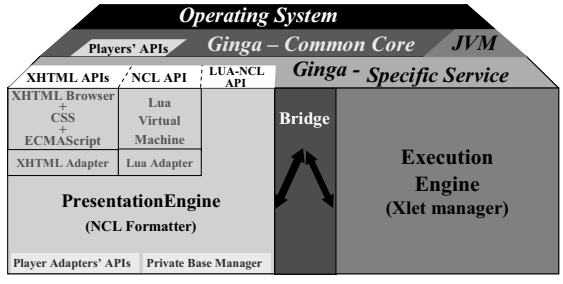

Figure 1: Ginga Archicteture.

Ginga-J is a logical subsystem of the Ginga System that processes Xlet object content. A key component of the procedural application environment is the procedural content execution engine, composed by a Java Virtual Machine. Ginga-J [9] is discussed in details in another paper of this same Journal Issue.

Ginga-NCL and Ginga-J are built over the services offered by the Ginga Common-Core module, whose organization is illustrated in Figure 2.

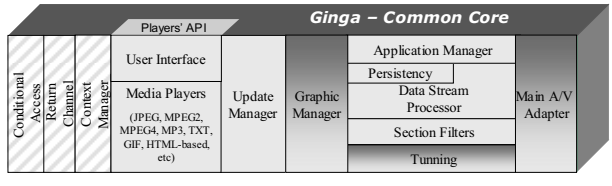

Figure 2: Ginga Common-Core.

Common content decoders serve both procedural and declarative application needs for the decoding and presentation of common content types such as PNG, JPEG, MPEG and other formats. The Ginga Common Core is composed by common content decoders and procedures to obtain contents, transported in MPEG-2 Transport Streams or via the return channel. The Ginga Common Core shall also support the conceptual display model specified for the Brazilian Terrestrial DTV [6].

The architecture and facilities of Ginga are intended to apply to broadcast systems and receivers for terrestrial (over-the-air) broadcast. However, the same architecture and facilities may be applied to other transport systems (such as satellite, cable TV, and IPTV systems).

In general, Ginga is unaware of any native applications that also may choose to share the graphics plane. These include but are not limited to: closed captioning, conditional access (CA) system messages, receiver menus, and native program guides. Native applications may take precedence over Ginga applications. Closed captioning and emergency

messaging shall take precedence over the Ginga System. Some native applications, such as closed captioning, present a special case where the native application may be active for long periods concurrently with Ginga applications.

3. GINGA-NCL

NCL (Nested Context Language) is the declarative language adopted by Ginga-NCL. Before presenting the reasons why NCL was chosen, a brief history comparing related work is important.

3.1. RELATED WORK

One of the first open standards used in DTV systems was defined by the ISO-MHEG (*Multimedia and Hypermedia Experts Group*) in 1997 [11]. The original specification offered a declarative approach for building multimedia applications, known as MHEG-1 (MHEG part 1), using ASN.1 notation to define object-based multimedia applications. It is worth to mention that in 1991, the Nested Context Model (NCM), the conceptual data model of NCL, proposed a solution to an open hypermedia issue [5], which was immediately adopted by MHEG as its composite structure, in the working group meeting of 1992. NCM solved the problem of composition nesting in structured documents, and took the name of the solution (nested context) [13].

Both NCM and MHEG-1 included support for objects that contained procedural code, which could extend their basic models to add decision-making features that were otherwise not possible. MHEG was overtaken by the portability success of Java, and thus, in 1998, it added support for using Java to develop script objects, thus mixing the declarative strengths of MHEG with the procedural elements of Java. Although this MHEG version (MHEG-6) was never deployed, it formed the basis of the DAVIC (Digital Audio Visual Council) standard for ITV, which had many of its API adopted by MHP (*Multimedia Home Platform*) [12].

MHP was the first open middleware standard based purely on Java. This shift from the declarative approach toward Java, due mainly to Java portability, does not mean that the declarative approach was left behind, however. For many applications, a declarative paradigm is perfectly adequate, and may actually be superior to Java.

Indeed, because they offer a higher abstraction level than Java, declarative languages are easier to use and does not required a deep programming knowledge from application authors. Hence, little by little the Java based middleware reincorporated the declarative environment. They have been included in MHP using HTML and plug-ins for non-Java application formats.

In addition, the great majority of datacasting applications (in particular those that handle media objects coming from not only the broadcast channel but also from the broadband return channel) deal with temporal synchronization of media objects, including user interaction as a particular case. As synchronization can be easily done using the declarative approach, the importance of declarative environments has become more and more evident.

All these facts have made the three main standards to adopt declarative environments: DVB-HTML [3], ACAP-X [1] and BML (*Broadcast Markup Language*) [2]. The last one, coming from the Japanese ISDB Standard, has indeed opted to only offer the declarative environment (BML) in its middleware. Although ISDB specifies an execution environment based on GEM (Globally Executable MHP) [4], its operation guidelines was never done.

Despite the importance of declarative environments, the focus of the declarative language of the three main mentioned standards is very poor when compared with the previous MHEG standard focus, as discussed in the next section. As stated in [15]: “MHEG were not very successful because the industry was not yet ready for the features offered by the standard”.

3.2. THE NCL LANGUAGE

All presentation engines of the three main DTV systems — BML [2], DVB-HTML [3] and ACAP-X [1] — use an XHTML-based language.

XHTML is a media-based declarative language, which means that the structure defined by the relationships among XHTML objects (XHTML documents or objects enclosed in XHTML documents) is embedded in the document’s media content. XHTML can thus be classified as a markup language: a formalism that describes a class of documents which employ markup in order to delineate the document’s structure, appearance and other aspects.

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 adaptability), are usually defined using imperative languages (e.g., ECMAScript); thus they cannot take profit of the easy authoring way offered in other declarative languages, like NCL and SMIL (*Synchronized Multimedia Integration Language*) [17].

Unlike HTML or XHTML, NCL and SMIL have a stricter separation between content and structure, and

they provide a non-invasive control of presentation linking and layout.

The focus of the NCL declarative language is broader than the XHTML counterpart. Generalized spatio-temporal synchronization, defined by NCL links; adaptability, defined by NCL switch and descriptor switch elements; and support for multiple exhibition devices, defined by NCL regions, are the focus of the NCL declarative language [7]. User interaction is treated just as a special case of temporal synchronization. The same facilities are also found in SMIL, but they are constrained to documents generated before the presentation begins. Live document generation is not supported by SMIL as it is in NCL, as discussed still in this section and in Section 4. Moreover, SMIL was still restricted to the Web and does not have support for a broadcast environment yet.

As NCL has a stricter separation between content and structure, NCL does not define any media itself. Instead, it defines the glue that holds media together in multimedia presentations. Therefore, an NCL document only defines how media objects are structured and related, in time and space. As a glue language, it does not restrict or prescribe the media-object content types. In this sense, we can have image objects (GIF, JPEG, etc.), video objects (MPEG, MOV, etc.), audio objects (MP3, WMA, etc.), text objects (TXT, PDF, etc.), execution objects (Xlet, Lua, etc.), etc., as NCL media objects. Which are the media objects supported depends on the media players that are embedded in the NCL formatter (NCL player), indeed the common media players offered by the Ginga Common Core module, as presented in Section 2. One of these players is, of course, the decoder/player implemented in hardware by the DTV receptor. In this way, note that the main video and audio stream is treated like all other media objects that can be related using NCL.

Another NCL media object that must certainly be supported is the HTML-based media object. Therefore, NCL does not substitute but embeds HTML-based documents (or objects). As with other media objects, which HTML-based language will have support in an NCL formatter is an implementation choice, and therefore it will depend on which HTML browser will act as a media player embedded in the NCL formatter.

As a consequence, it is possible to have BML browsers, DVB-HTML browsers and ACAP-X browsers embedded in an NCL document player. It is even possible to have all of them. It is also possible to receive a browser code through datacasting and install it as a plug-in (usually a Java plug-in).

It must be stressed that, in order to support all XHTML-based browser facilities defined by other DTV standards, all Brazilian Terrestrial DTV specifications related to datacasting must also support these facilities, such as the transport of stream events (see Section 5), for example.

Although an XHTML-based browser must be supported, the use of XHTML elements to define relationships (including XHTML links) must be dissuaded when authoring NCL documents. Structure-based authoring must be emphasized for the well-known reasons largely reported in the literature.

During the exhibition of media-object contents, several events are generated. Examples of events are the presentation of marked segments of media-object content, the selection of a marked content segment, etc. Events may generate actions on other media objects, like to start or stop their presentations. Hence, events must be reported by media players to the NCL formatter that, in its turn, can generate actions to be applied to these or other players. Ginga-NCL defines an adapter API [7] to standardize the interface between the Ginga-NCL formatter, and each specific player, as shown in Figure 1.

When any media player, in particular an XHTML-based browser, is integrated to the Ginga-NCL formatter, it must support the adapter API. For some media players, including XHTML-based browsers, an adapter module may be necessary to accomplish the integration.

Finally, for live editing, Ginga-NCL has also defined NCL stream events in order to support live generated events in stream media, in particular the main program stream video. These events are a generalization of the same concept found in other standards, like for example the b-events of BML. Again, although an XHTML-based browser must be supported, the use of XHTML elements to define relationships (including stream events) must be dissuaded in authoring NCL documents, for the same motivation: structure-based authoring must be emphasized.

For all these aforementioned reasons, NCL has become the natural choice for the Brazilian DTV standard. Its detailed specification using XML Schemas can be found in [7].

3.3. NCL Media Objects

As aforementioned, NCL does not restrict or prescribe the media-object content types. In this sense, we can have image objects, video objects, audio objects, text objects, and also XHTML-based objects, which may contain script objects embedded. In the case of the Ginga-NCL middleware, the unique script code allowed

to be embedded in XHTML-based object is ECMAScript code.

NCL also allows media objects whose content is a procedural code specified in some procedural language. However, the Ginga-NCL scripting language is Lua, as briefly discussed in Section 6. Ginga-NCL also offers support to media objects carrying XLet codes, as discussed in Section 7.

3.4. NCL Language Elements: An Overview

NCL is an XML application that follows the modularization approach. The modularization approach has been used in several W3C language recommendations. A *module* is a collection of semantically-related XML elements, attributes, and attribute's values that represents a unit of functionality. Modules are defined in coherent sets. A *language profile* is a combination of modules. For the Brazilian Terrestrial DTV, Ginga-NCL defines two language profiles: the EDTVProfile (*Enhanced Digital TV Profile*) and the BDTVProfile (*Basic Digital TV Profile*).

This section briefly describes the main definitions made by Ginga-NCL. The complete definition of the NCL 3.0 modules, using XML Schemas, is presented in [7]. Any ambiguity found in this text may be clarified by consulting the XML Schemas.

The basic NCL structure module defines the root element, called <ncl>, and its children elements, the <head> element and the <body> element, following the terminology adopted by other W3C standards.

The <head> element can have <importedDocumentBase>, <ruleBase>, <transitionBase> <regionBase>, <descriptorBase>, <connectorBase>, <meta>, and <metadata> elements as its children.

The <body> element can have <port>, <attribute>, <media>, <context>, <switch>, and <link> elements as its children. The <body> element is treated as an NCM context node. In NCM [14], the conceptual data model of NCL, a node can be a context, a switch or a media object. Context nodes can contain other NCM nodes and links. Switch nodes contain other NCM nodes. NCM nodes are represented by corresponding NCL elements.

The <media> element defines a media object specifying its type and its content location. The several types are those discussed in Section 3.3, besides the "application/x-ginga-settings" type, specifying an object whose attributes are global variables defined by the document author or are reserved environment variables that can be manipulated by the NCL document processing; and the "application/x-ginga-time" type, specifying a special <media> element whose content is the Greenwich Mean Time (GMT).

The <context> element is responsible for the definition of context nodes. An NCM context node is a particular type of NCM composite node and is defined as containing a set of nodes and a set of links, as aforementioned. Like the <body> element, a <context> element can have <port>, <attribute>, <media>, <context>, <switch>, and <link> elements as its children.

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

The NCL Interfaces functionality allows the definition of node interfaces that are used in relationships with other node interfaces. The <area> element allows the definition of content anchors representing spatial portions, temporal portions, or temporal and spatial portions of a media object (<media> element) content. The <port> element specifies a composite node (<context>, <body> or <switch> element) port with its respective mapping to an interface of one of its child components. The <attribute> element is used for defining a node attribute or a group of node attributes as one of the node's interfaces. The <switchPort> element allows the creation of <switch> element interfaces that are mapped to a set of alternative interfaces of the switch's internal nodes.

The <descriptor> element specifies temporal and spatial information needed to present each document component. The element can refer a <region> element to define the initial position of the <media> element (that is associated with the <descriptor> element) presentation is some output device. The definition of <descriptor> elements must be included in the document head, inside the <descriptorBase> element, which specifies the set of descriptors of a document. Also inside the document <head> element, the <regionBase> element defines a set of <region> elements, each of which may contain another set of nested <region> elements, and so on, recursively; regions define device areas (e.g. screen windows) and are referenced by <descriptor> elements, as previously mentioned.

A <causalConnector> element represents a relation that can be used for creating <link> elements in documents. In a causal relation, a condition must be

satisfied in order to trigger an action. A <link> element binds (through its <bind> elements) a node interface with connector roles, defining a spatio-temporal relationship among NCL objects (represented by <media>, <context>, <body> or <switch> elements).

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

In order to allow an entity base to incorporate another already-defined base, the <importBase> element can be used. Additionally, an NCL document can be imported through the <importNCL> element. The <importedDocumentBase> element specifies a set of imported NCL documents, and must also be defined as a child element of the <head> element.

Some important NCL element's attributes are defined in other NCL modules. The EntityReuse module allows an NCL element to be reused. This module defines the *refer* attribute, which refers to an element URI that will be reused. Only <media>, <context>, <body> and <switch> can be reused. The KeyNavigation module provides the extensions necessary to describe focus movement operations using a control device like a remote control. Basically, the module defines attributes that can be incorporated by <descriptor> elements.

Some SMIL functionalities are also incorporated by NCL. The <transition> element and some transition attributes are defined in the SMIL BasicTransitions module and the SMIL TransitionModifiers module. The NCL <transitionBase> element specifies a set of transition effects, defined by <transition> elements, and must be defined as a child element of the <head> element.

Finally, the SMIL MetaInformation module is also incorporated. Meta-information does not contain content information that is used or display during a presentation. Instead, it contains information about content that is used or displayed. The Metainformation module contains two elements that allow describing NCL documents. The <meta> element specifies a single property/value pair. The <metadata> element contains information that is also related to meta-information of the document. It acts as the root element of an RDF tree: RDF element and its sub-elements (for more details, refer to W3C metadata recommendations [16]).

4. GINGA-NCL EDITING COMMANDS

The core of the Ginga-NCL presentation engine is composed by the NCL Formatter and the Private Base Manager. The NCL Formatter is in charge of receiving an NCL document and controlling its presentation, trying to guarantee that the specified relationships among media objects are respected. The formatter deals with NCL documents that are collected inside a data structure known as *private base*. Ginga associates a private base with a TV channel.

The Private Base Manager is in charge of receiving NCL document editing commands and maintaining the active NCL documents (documents being presented). XML-based *NCL editing commands* [7] are divided in three subsets.

The first subset focuses on the private base activation and deactivation (openBase, activateBase, deactivateBase, saveBase, and closeBase commands).

NCL documents can be added to a private base and then can be started, paused, resumed, stopped and removed, through well defined commands that compose the second subset.

The third subset defines commands for live editing, allowing NCL elements to be added and removed, and allowing values to be set to NCL <attribute> elements. *Add* commands always have NCL elements as their arguments. Whether the specified element already exists or not, document consistency must be maintained by the NCL formatter, in the sense that all element attributes classified as required must be defined. The elements are defined using a XML-based syntax notation identical to that used by the NCL document definition (defined by NCL schemas).

The DSM-CC is adopted in Ginga for carrying editing commands in MPEG-2 TS elementary streams. DSM-CC stream events and DSM-CC object carousel protocol are discussed in the next section.

5. GINGA COMMON CORE DATA PROCESSING

As briefly introduced in Section 2, the Ginga Common Core is composed by common content decoders and shall support the conceptual display model specified for the Brazilian Terrestrial DTV [6]. Procedures to obtain contents, transported in MPEG-2 Transport Streams or via a return channel, and to rebuild the data structure required by the NCL document processing is also a function of the Common Core. These procedures are the focus of this section.

The DSM-CC is adopted in Ginga for carrying editing commands in MPEG-2 TS elementary streams.

DSM-CC stream events and the DSM-CC object carousel protocol are the basis for document handling in the Ginga presentation engine.

As usual, DSM-CC stream-event descriptors must have a structure composed basically by an id (identification), a time reference and a private data field. The identification uniquely identifies the stream event as an editing command. The time reference indicates the exact moment when to trigger the event. A time reference equal to zero informs that the event must be triggered immediately after being received (events carrying this type of time-reference are commonly known as "do it now" events). The private data field provides support for event parameters; in the case of editing command stream event descriptor, the editing-command parameters. If an XML-based *NCL command parameter* is short enough, it is transported directly in the stream event descriptor payload. Otherwise, the payload carries only references to the XML-based command parameter (an XML file and its associated media content files, if necessary) that is put in file systems to be transported by DSM-CC object carousels.

The DSM-CC object carousel protocol allows the cyclical transmission of event objects and file systems. A DSM-CC carousel generator is used to join the file systems and stream event objects into an elementary stream.

Event objects are used to map stream event names into stream event ids. Event objects are used to inform Ginga about DSM-CC stream events it should receive. Event names allow specifying types of events, offering a higher abstraction level for middleware applications when registering and handling DSM-CC stream events. The Private Base Manager, as well as NCL execution-objects (e.g. NCLua, NCLet, see next two sections), must use event names to register themselves as listeners of stream events they handle. The Private Base Manager must register itself as listener of the "*gingaEditingCommand*" stream event name.

The transmission of NCL Document files can be done through an object carousel or through an interactive channel protocol [10].

When XML-based *NCL command parameters* (NCL documents or NCL nodes to be included in a NCL document) are transmitted through an object carousel, the Ginga Common Core is responsible for mounting each file system organization that composes the XML-based parameter at the receiver device. In the same object carousel that carries the NCL document or node specification, an event object must be transmitted in order to map the name "*gingaEditingCommand*" to the eventId of the DSM-CC stream event descriptor, which shall carry the corresponding editing command. The stream event descriptor shall carry a set of reference pairs {uri, ior}. The uri parameter of the first pair must have the "x-isdtv" schema and the local absolute path of the NCL document or the NCL node specification (the path in the data server). The corresponding ior parameter in the pair must reference the NCL Document or NCL Node specification IOR (carouselId, moduleId, objectKey) in the object carousel [8]. If other file systems must be transmitted using other object carousels in order to complete the addDocument or addNode command with media content, other {uri, ior} pairs must be present in the command. In this case, the uri parameter must have the "x-isdtv" schema and the local absolute path of file system root (the path in the data server), and the corresponding ior parameter in the pair must reference the IOR of any root-child file or directory in the object carousel (the IOR of the carousel service gateway). Figure 3 depicts an example of an NCL document transmission through an object carousel.

In this example, a content provider wants to transmit an interactive program named "interactiveProgram.ncl" stored in one of its data servers (Local File System, in Figure 3). An object carousel must then be generated (Service Domain = 1, in Figure 3) carrying all the interactive program contents (the NCL file and all media files) and also an event object (moduleId = 2 and objectKey = 2, in Figure 3) mapping the "*gingaEditingCommand*" name to the eventId value (value "3", in Figure 3). A stream event descriptor must also be transmitted with the appropriated eventId value, in the example "3", and the commandTag value specifying an addDocument command (hexadecimal code 0x05). The uri parameter shall have the "x-isdtv" schema and the absolute path of the NCL document ("C:\nclRepository\weather", in Figure 3). Finally, the IOR of the NCL document in the object carousel is carried in the xmlDocument parameter (carouselId = 1, moduleId = 1, objectKey = 2, in Figure 3).

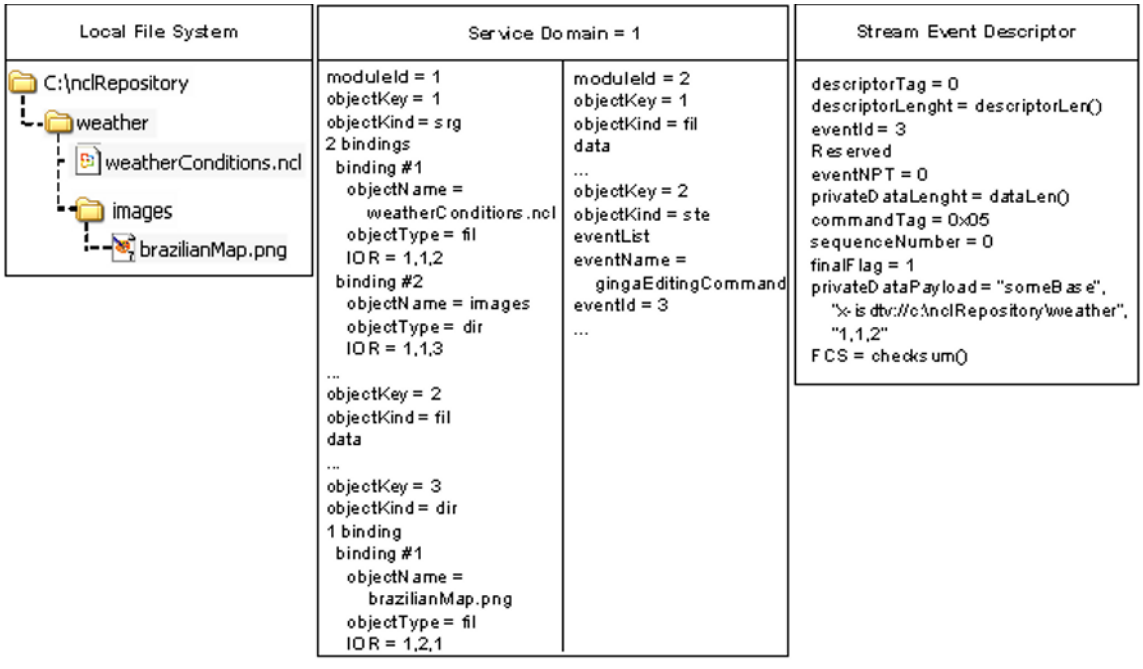

Figure 3: Example of an NCL Document Transmission.

On the other hand, if an interactive channel protocol is used to download an NCL document through the return channel, the download must be requested through the document *uri* parameter passed in the *addDocument* editing command. In addition, at least one program related object carousel must be transmitted, carrying an event object mapping the name "*gingaEditingCommand*" to the eventId of the DSM-CC stream event, which shall carry the *addDocument* editing command.

6. LUA: THE NCL SCRIPT LANGUAGE

The scripting language adopt by Ginga-NCL to implement procedural objects embedded in NCL documents is *Lua* (<media> elements of type "application/x-ginga-NCLua").

NCLua provides an `nclCommand` library that offers a set of functions to support NCL's editing commands, with the same semantics described in Section 4, with one additional facility. All functions can receive a time reference as an optional parameter that can be used to specify the exact moment when the editing command should be executed. This parameter has the same scheduling function of the time reference present in stream events (see Section 5).

If the time-reference optional parameter is not provided in the function call, the editing command should be executed immediately. When provided, this parameter can have two different types of values, with different meanings. If it is a `number` value, it defines the amount of time, in seconds, for how long the command must be postponed. However, this parameter can also specify the exact moment to execute the command in terms of absolute values. For that, this parameter must be a `table` value with the following fields: `year` (four digits), `month` (1-12), `day` (1-31), `hour` (0-23), `min` (0-59), `sec` (0-61), and `isdst` (daylight saving flag, a `boolean`).

NCLua also provides an `nclExtended` library that offers a set of functions to start, stop, pause or resume an interface defined in NCL for NCLua media object executions. These function results can be used as conditions, evaluated by the NCL formatter, to trigger actions on other NCL objects of the same document. Besides commanding event state transitions, a procedural NCLua code can also register itself as a listener of state transitions of any NCL event.

Finally, depending on the middleware configuration, it is possible to have access in Lua to the same API provided by the Ginga-J, the Ginga procedural environment, in order to have access to some set-top box resources and Ginga facilities. The API provided in Lua must follows the same specification presented in [8], and discussed in another paper of this same Journal issue.

7. THE BRIDGE BETWEEN GINGA-NCL AND GINGA-J

The two-way bridge between Ginga-NCL and Ginga-J is done:

- in one way, through NCL relationships, defined in <link> elements that refers to <media> elements representing Xlet ("application/x-ginga-NCLet" type) codes supported by Ginga-J; and through Lua scripts (<media> elements of the "application/x-ginga-NCLua" type) referencing Ginga-J methods;
- in the reverse way, through Ginga-J functions that can monitor any NCL events and can also command changes in NCL elements and attributes, through relationships defined in <link> elements or through NCL editing commands, similar to what is done by NCLua objects, as discussed in Section 6.

8. FINAL REMARKS

This paper has discussed the main features of the declarative environment of the Brazilian Terrestrial DTV standard. The discussion has been based on the Ginga architecture, in particular the Ginga-NCL functionalities.

Different from the other main middleware systems, the declarative environment is based on a structured-based language, the NCL language. NCL is unique in the sense that:

- it is structure-based;
- it provides support for spatio-temporal synchronization relationship specifications, including but not being restricted to user interaction based relationships;
- it provides support for alternative content and presentation definition;
- it provides support for exhibition on multiple simultaneous devices;
- it can manipulate global and local variables, through NCL entities or through procedural objects;
- it does not restrict or prescribe the media-object content types, and thus it does not substitute but can embed HTML-based documents, including those defined by BML, DVB-HTML and ACAP-X documents, depending only on the browser implementation;
- it allows live editing through stream event commands;
- it provides support for objects that contain procedural code (NCLua and NCLet), which can extend NCL basic model to add decision-making features that were otherwise not possible; and
- it provides a powerful scripting language.

No other declarative language offers such a set of characteristics so important in a DTV domain.

As for the scripting language, Lua has also a very particular set of characteristics that makes it unique in a DTV domain:

- it has a simple procedural syntax with powerful data description constructs;
- it is dynamically typed;
- it has automatic memory management with garbage collection, making it ideal for configuration, scripting, and rapid prototyping;
- it has also other features that make it a good option for embedded and soft real-time systems, as is the case of DTV systems: incremental garbage collection, weak references and co-routines;
- it is interpreted from bytecodes;
- it is an "embeddable" language: the interpreter is a pure ANSI C library;
- it is small, and as consequence, easy to install, easy to adapt and portable;
- it is efficient (faster than Perl and Python; much faster than JavaScript).

All these features made Lua very well accepted by game developers and favor the language success as a scripting language in the DTV domain.

Also different from the other main middleware systems, the Ginga's declarative environment and procedural environment has been conceived to be an integrated system from the beginning of its design. Therefore there is no function duplication and neither any patch, what makes Ginga light and efficient, despite of its great expressiveness.

REFERENCES

[1] Advanced Application Platform (ACAP), *ATSC Standard: Document A/101*, August 2005.

[2] ARIB B-24 XML-based Multimedia Coding Scheme, *ARIB Standard B-24 Data Coding and Transmission Specifications for Digital Broadcasting*, version 4.0, February 2004.

[3] P. Perrot. DVB-HTML - An Optional Declarative Language within MHP 1.1, *EBU Technical Review*. 2001.

[4] Digital Video Broadcasting (DVB) Globally Executable MHP (GEM) Specification 1.0.0. *ETSI TS 102 819 V1.1.1*. Available at http://pda.etsi.org/pda/home.asp?wki_id=17842

[5] F.G. Halasz. Reflexions on Notecards: Seven Issues for the Next Generation of Hypermedia Systems. *Communications of ACM*, Vol.31, No. 7. July 1988.

[6] Volume 1 Standard 06. *Brazilian DTV Fórum*. April 2007.

[7] Volume 2 Standard 06. *Brazilian DTV Fórum*. April 2007.

[8] Volume 3 Standard 06. *Brazilian DTV Fórum*. April 2007.

[9] Volume 4 Standard 06. *Brazilian DTV Fórum*. April 2007.

[10] Standard 07. *Brazilian DTV Fórum*. December April 2007.

[11] ISO/IEC 13522-5. Information Technology – Coding of multimedia and hypermedia information – Part 5: Support for base-level interactive applications. *ISO Standard*, 1997.

[12] ETSI. Digital Video Broadcasting (DVB), Multimedia Home Platform (MHP) Specification 1.1.1, *ETSI TS* 102 812, 2003.

[13] M.A. Casanova, L. Tucherman, M.J. Lima, J.L. Rangel, N.R. Rodriguez, L.F.G. Soares. The Nested Contex Model for Hyperdocuments. *Proceedings of Third ACM Conference on Hypertext*, San Antonio, Texas. Dezembro de 1991; pp. 193-201.

[14] L.F.G Soares; R.F Rodrigues. Nested Context Model 3.0: Part 1 – NCM Core, *Technical Report, Departamento de Informática PUC-Rio*, May 2005, ISSN: 0103-9741. Also available in www.telemidia.puc-rio.br/maestro.

[15] S. Morris, A. S. Chaigneau. Interactive TV Standards. *Focal Press*, Elsevier, 2005.

[16] O. Lassila, R. R. Swick. Resource Description Framework (RDF) Model and Syntax Specification, *W3C Recommendation*, 22 February 1999. Available at http://www.w3.org/TR/REC-rdf-syntax/.

[17] SMIL 2.1 - Synchronized Multimedia Integration Language – SMIL 2.1 Specification*, W3C Recommendation*, December 2005.