Adaptable software components in an electronic program/service guide application architecture for context aware guide presentation

Marcio Ferreira Moreno*

Departamento de Informática – PUC-Rio,
Rua Marquês de São Vicente,
225 Rio de Janeiro/RJ, 22453-900, Brazil
E-mail: mfmoreno@inf.puc-rio.br
*Corresponding author

Carlos de Salles Soares Neto

Departamento de Informática – UFMA,
Av. dos Portugueses, Campus do Bacanga,
São Luís/MA, 65080-040, Brazil
E-mail: csalles@deinf.ufma.br

Luiz Fernando Gomes Soares

Departamento de Informática – PUC-Rio,
Rua Marquês de São Vicente,
225 Rio de Janeiro/RJ, 22453-900, Brazil
E-mail: lfgs@inf.puc-rio.br

Abstract: This paper aims at providing architecture to create Electronic Program and Service Guide applications that can be adapted in real time. The architecture can be applied to generate computational code in different target languages, with a clear separation between style, structure and navigational aspects of Electronic Program and Service Guides. Following the proposed architecture and serving as a proof of concept, an Electronic Program Guide (EPG) was implemented for the reference implementation of the declarative environment of Ginga, the middleware of the Brazilian Digital TV System.

Keywords: Ginga-NCL; middleware; DTV; NCL; nested context language; EPG; electronic program guide.

Reference to this paper should be made as follows: Moreno, M.F., Soares Neto, C.S. and Soares, L.F.G. (2009) 'Adaptable software components in an electronic program/service guide application architecture for context aware guide presentation', *Int. J. Advanced Media and Communication*, Vol. 3, No. 4, pp.351–364.

Biographical notes: Marcio Ferreira Moreno (Rio de Janeiro, Brazil, 1979) received BS Degree from Department of Informatics, Federal University of Juiz de Fora (UFJF), Brazil in February 2004. He received his MS Degree from Department of Informatics, Catholic University of Rio de Janeiro (PUC-Rio),

Brazil in April 2006. Currently he is a PhD student in PUC-Rio, Brazil. He is associate researcher on TeleMídia Lab, PUC-Rio, and has worked on terrestrial Brazilian DTV specifications and Ginga-NCL reference implementation. His main interests are DTV systems, DTV middlewares and multimedia and hypermedia systems and networks.

Carlos de Salles Soares Neto (São Luís, Brazil, 1978) received BS Degree from Department of Informatics, Federal University of Maranhão (UFMA), Brazil in August 2000. He received his MS Degree from Department of Informatics, Pontifical Catholic University of Rio de Janeiro (PUC-Rio), Brazil in August 2003. Currently he is a PhD student in PUC-Rio, Brazil. He is an Assistant Professor at Federal University of Maranhão (UFMA) and associate researcher on TeleMídia Lab, PUC-Rio, and on LAWS, UFMA (Laboratory of Advanced Web Systems). His main interests are the authoring process of DTV applications and multimedia development tools.

Luiz Fernando Gomes Soares is a full Professor at the Informatics Department in the Catholic University of Rio de Janeiro (PUC-Rio), where since 1990 he heads the TeleMidia Lab. He is a Board Member of the Brazilian Internet Steering Committee and Chair of the Middleware Brazilian Working Group for the Brazilian Digital TV System. Prior to join PUC-Rio he was a researcher at the Brazilian Computer Company. Other academic appointments include visiting professorships in the computer science at École Nationale Superieure de Télécomunications (France), Université Blaise Pascal (France), and Universidad Federico Santa Maria (Chile).

1 Introduction

New encoding and digital compression techniques have made possible a considerable proliferation of media contents. Digital TV takes profit of this technical evolution offering contents (programmes and services) both received by broadcasting and on demand. With so many content options, it is essential to have applications able to gather information about these contents (usually called metadata or meta-information) and to present it in an organised and adaptable way to viewers. In a digital TV system, among these metadata are the number and the name of the content provider (a TV channel number and name, for example), and the name, description, start time and duration of each content (also called event).

In the literature, applications that manage and present such metadata are divided into two types: EPGs, which compiles information received by broadcasting (Morris and Smith-Chaigneau, 2005), and Electronic Service Guides (ESGs), which in addition includes content information that can be acquired on demand (Hjelm, 2008).

Typically, in a terrestrial broadcasting system for interactive digital TV there is a standardised format for metadata and they are sent multiplexed with other contents (audio, video and data) of schedule information (Morris and Smith-Chaigneau, 2005). Metadata about content received on demand, as in IPTV, P2PTV and even DTV through using its interactive (return) channel, usually follow a format defined or adopted by the service provider.

An EPG/ESG can be developed as a receiver's resident application or as an application pushed by broadcasting or pulled on demand. It can mirror metadata

transmitted from all broadcast stations and service providers, or it may simply reflect metadata of a specific service, as for example, a tuned channel in a DTV system. Here, we must make a difference between the EPG/ESG application, from now on called Electronic Guide Application (EGA), and the result of its execution, the EPG/ESG information to be presented, from now on called Electronic Guide Information (EGI).

Generally, an EGA presents metadata information adapted to user and receiver profiles. Not only the information to be presented can be adapted but also the layout used in the presentation. However, hardly ever the EPG/ESG application itself is adaptable. Sometimes, it is just possible that it suffers sporadic updates, but not to be adapted on-the-fly (*adaptive EGA*). In other words, the EGI presented is usually adaptable, but hardly ever the EGA that generates the EGI. The lack of adaptability makes it difficult or unviable to reuse same components of an EGA to offer different algorithms to create presentation and navigational layout depending on the logged or tuned service provider. As an example, in a DTV system it would be difficult or impossible to reuse the same EGA for different presentations depending on which TV channel is tuned; as a result, each broadcaster would have to implement its own application as a whole (without sharing components with other broadcasters). It must be noted, however, that when an EGA is adapted, usually its resultant EGI also suffers adaptations. Let us take the last DTV example, changing EGAs when TV tuning is changed may result in different (adapted to the tuned channel) EGIs.

Aiming at providing adaptive EGAs, this paper presents a modular EGA architecture, including a meta-service responsible for EGAs dynamic adaptations. The resulting adaptive EGAs will be able to build a metadata presentation adapted to user and receiver profiles (adaptive EGIs).

The proposed architecture is hybrid with respect to the location of its modules: some components can be resident and some others can be received by broadcast or on demand. The general purpose proposed architecture could be a solution for terrestrial digital TV systems, IPTV, P2PTV etc. As a proof of concept, an adaptive EPG implementation has been carried out using the Nested Context Language (NCL) declarative language and its Lua scripting language (ABNT NBR 15606-2, 2007). This EPG was incorporated in the reference implementation of the declarative middleware of the Brazilian Terrestrial Digital TV System (SBTVD-T), named Ginga-NCL.[1] The implementation takes profit of the easy way provided by Ginga-NCL to build adaptive declarative applications by using NCL live editing commands (ABNT NBR 15606-2, 2007), and provided by NCL language elements.

The remainder of this paper is organised as follows. Section 2 discusses related works. Section 3 presents the EGA architecture in its main modules. In conformance to this architecture, Section 4 describes an EPG implementation integrated with the Ginga-NCL middleware. Finally, Section 5 is reserved for the final remarks.

2 Information and support provided by TV systems-related works

As aforementioned, an EGA is in charge of interpreting metadata information and present them, through an EGI, in a way that human beings can understand and access. There are, at least, two adaptable elements involved in this process: the way metadata are acquired, interpreted processed and presented (which compound an EGA architecture); the metadata structure created in the process itself (through which the EGI is generated).

To provide metadata, the main broadcast Digital TV systems (Morris and Smith-Chaigneau, 2005) define a set of tables called Service Information (SI) on European DVB (Digital Video Broadcast) (DVB Document A038 Rev. 3, 2007), Japanese Association of Radio Industries and Businesses (ARIB) (ARIB STD-B10, 2006) and Brazilian SBTVD (Digital TV System of Brazil) (ABNT NBR 15603-1, 2007) standards and Program and System Information Protocol (PSIP) in the American Advanced Television System Committee (ATSC) standard (ATSC A/65b, 2003).

Among these tables, there is a specific one, named Event Information Table (EIT), responsible for carrying metadata of the schedule information, i.e., events that make up the schedule. Each event consists of a unique identifier, a starting time, its duration and a set of descriptors for additional information (DVB Document A038 Rev. 3, 2007; ARIB STD-B10, 2006; ABNT NBR 15603-1, 2007; ATSC A/65b, 2003). EIT tables are encapsulated and multiplexed with other contents, generating a single stream transmitted by broadcast to the receivers.

In IPTV and P2PTV systems metadata commonly come from many sources: service providers, content providers, etc. (Hjelm, 2008). In this scenario, there are a significant number of possible metadata standards that can be used for building EPG/ESGs. These metadata can be sent to the receiver pushed by a multicast service; or it can be pulled on demand.

The vast majority of EPG/ESG related work concerns only to adaptive EGI, which is real time adaptations of the metadata structure to be presented.

Smyth and Cotter (2001), Ardissono et al. (2003) and Vaguetti and and Gondim (2008) have developed receiver's recommendation applications in charge of generating personalised EGIs based on user profiles.

O'Sullivan et al. (2004) use data mining techniques and user profile characterisation to provide an EPG with schedule recommendations that guide viewers to meet faster their preferences. In practice, the proposed architecture allows the inclusion of a recommendation system as part of an EGA to adapt the EGI, but nothing is mentioned regarding changing the algorithms and techniques used.

When EGAs can be adapted, the vast majority of systems does allow only minor adjustments (adaptations). Either they are resident applications, or they are received through a network. However, they can only be replaced, but not adapted on-the-fly.

As an example, an open source project called MythTV[2] aims at developing resident applications to transform a personal computer into a Personal Video Recorder (PVR). The goal is to adapt personal computers to offer functionalities as media player, DVD burning, internet browsing, remote control handling, etc. MythTV defines an EPG as a resident application, without any facilities for runtime adaptations, although it is possible to modify its open source code before compile time.

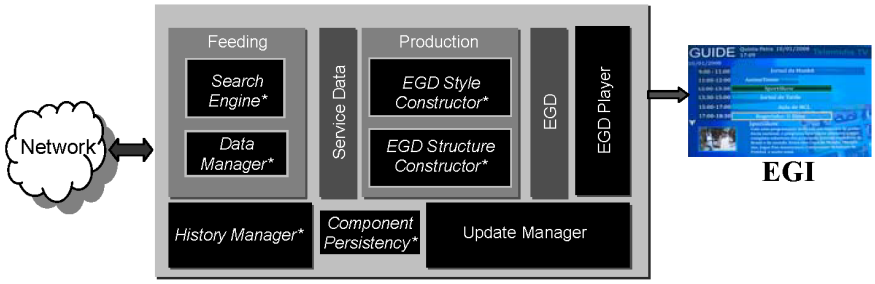
The architecture proposed in this paper aims at providing an adaptive EGI both providing support for metadata structure adaptations (as offered in all mentioned related work) as well as providing support for runtime adaptation on EPG/ESG software components.

3 EGA architecture

The EGA architecture is divided in eight components, as presented in black boxes in Figure 1. With the exception of the Update Manager component, the meta-service

in charge of other component updates and the Electronic Guide Document (EGD) player, all other six components can be adapted in real time.

Figure 1 EGA architecture (see online version for colours)



The entire EGI generating process is divided into feeding and production. In the feeding stage, the meta-information to be processed and presented in the EGI are obtained and transformed into internal data structures named Service Data, which are then passed to the production stage. During production, another data structure, called Electronic Guide Document (EGD) is built for presentation by the EGD player, usually a receiver middleware language engine.

The Feeding process is performed by transforming meta-information pushed by broadcasters or obtained on demand from service providers (through an interactive channel or through an IP network, for example). The Search Engine component, controlled by the Data Manager component, is responsible for gathering this information step by step. In each step, once the information is gathered, the Data Manager begins its translation process adding the result to the Service Data structure. During the translation process, context information (viewer preferences, for example) can be checked to see if the information gathered by the Search Engine is relevant.

The Service Data structure is a list of processed service information in a special internal EGA format. It represents information received about events, which can already be enriched by additional content obtained by the Search Engine. This information is available to the production process and also to the History Manager component. It is through the Service Data that the feeding and production stages are decoupled, allowing the replacement of any feeding component in real time.

The History Manager can make the data collected from the Data Service persistent, for future use by other applications. For example, they can be used by recommender systems in Social TV scenarios.

The output resulting from the production process is a set of instructions, in some TV middleware supported language: the EGD. The Production process separates relevant aspects of the EGD layout and the EGD data presentation structure. The EGD Style Constructor component is responsible for creating the presentation styles to be used when displaying the information extracted from the Service Data. The EGD Structure Constructor component is responsible for organising, structuring and relating contents to be presented.

For each middleware supported language (imperative or declarative language) a production module (EGD Style Constructor and EGD Structure Constructor) is necessary if the EGD is to be exhibited using this language engine. This point will be evident in the NCL implementation discussion.

The EGD is processed by the EGD player (usually a middleware language engine) that then displays the desired EGI. Note that EGI adaptations can also be performed by the EGD Player during its processing. Indeed adaptations in EGI may occur when the service data is generated by the Feeding process, when the EGD is created by the production process and finally when the EGD is played.

The update manager is a meta-service responsible for maintaining standard versions (default) of all mentioned modules (except the EGD player) and to receive requests to update them in real time. For each updated component, a validation process is carried out to ensure that standardised interfaces are met and that the system consistency is maintained. Updates can be done using persistent components already present in the Update Manager, or using components received through some network. Security and access right issues must also be handled by the Update Manager before updates take place.

Finally, the component persistency is responsible for making persistent any received up-to-date component.

3.1 Integrating EGA architecture to a middleware environment

Usually a middleware environment is composed of a language engine that takes care of an application life cycle, as well as a module to look after command events that can alter the application life cycle.

The EGA architecture, except for its EGD Player, is independent from the middleware platform but it is easier to be implemented in a platform that supports interpreted languages due to the facilities they provide to dynamic code changes.

Thus, declarative middlewares (also called presentation engines) provide good support for the proposed architecture implementation. In this case, both EGA (all components) and EGD can be implemented as declarative language documents. User agents take care of the EGA document execution that results in an EGD to be presented by the same or another user agent instantiation. The EGA document should be able to be adapted in real time by command events external to the user agent.

Any XHTML-based middleware can be taken as an example of the proposed architecture integration. To briefly illustrate the concept let us take the BML (ARIB Standard STD-B24, 2005) declarative middleware. In this case, the EGA can be an XHTML document embedding several ECMAScript objects implementing the several previously described components, which can be resident or received (or adapted) via one of the receiver's supported networks. The XHTML EGA can receive BML b-event commands (ARIB Standard STD-B24, 2005) triggering Update Manager's ECMAScript functions that take care of component substitution (stopping the current component ECMAScript code and starting the received new one). The result of the XHTML EGA execution is another XHTML document (the EGD) also played by the BML presentation engine.

Another example of integration in presentation engine with better declarative support can be given using the Ginga-NCL middleware. In this case, not only the EGA components compound an NCL application (NCL EGA), but also the resultant EGD may be another NCL document (NCL EGD, embedded in the EGA document). The first one can be adapted taking profit of the easy way Ginga-NCL provides to build adaptive declarative applications by using NCL live editing commands

(ABNT NBR 15606-2, 2007). The second one can be adapted by using NCL facilities provided by its language elements (Soares and Rodrigues, 2006).

Soares and Rodrigues (2006), including its scripting language Lua (Ierusalimschy, 2008), is the declarative language of Ginga (ABNT NBR 15606-2:2007, 2007). NCL has several facilities to support an adaptive NCL EGA and an adaptive EGD definition:

- A very well-defined separation between a document layout (defined in NCL <region> and <descriptor> elements) and a document structure (defined in NCL <media> and <context> elements). This will easy the EGD Style Constructor and the EGD Structure Constructor implementation, since there is no fusion of concepts.
- A good declarative support to alternative choices, both among contents and among different forms to present a same content (defined in NCL <switch> and <descriptorSwitch> elements, respectively). This will allow EGI runtime adaptations (performed by running the EGD) with regards to a user or a receiver profile; will allow EGA components real time updates and runtime adaptations depending on the service provider and user profiles.
- A very accurate support to define relationships among NCL content objects (defined both in NCL <link> elements and in NCL key navigational attributes). This will allow interactions between NCL EGA components and will allow navigation specification on NCL EGD contents.
- An option to embed imperative code and declarative code as an NCL object (as NCL <media> element's content). This will allow embed Lua script components in an NCL document, performing each EGA components that can be updated. This will also allow to embed an NCL EGD or a SMIL (2008) EGD into an NCL EGA document.
- The possibility of using multiple devices (defined in NCL <regionBase> elements) to display each NCL media content (including objects with NCL and SMIL[3] code). This will allow, for example, displaying an NCL EGD in a set of exhibition devices and another SMIL EGD version in another set of exhibition devices. It must be remarked that for each different version of EGD different EGD style constructor and EGD structure constructor must be implemented.

Besides the several aforementioned NCL facilities, Ginga-NCL (the presentation engine of Ginga) offers a good support to a well-defined set of live editing commands (ABNT NBR 15606-2, 2007). An NCL EGA implementation will take profit of this support both to receive commands that trigger updates in any of its components and also to build the NCL EGD step by step,[4] in real time. Indeed, the EGD will be an adaptive document also in the sense that it will be modified on-the-fly as soon as new event meta-information arrives.

The core of the Ginga-NCL presentation engine is composed of the NCL formatter and the Private Base Manager module. 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. Thus, an NCL Formatter instantiation plays the role of the EGD Player component.

The formatter deals with NCL documents that are collected inside a data structure known as private base. The Private Base Manager is in charge of receiving NCL editing commands and maintaining the active NCL documents (documents being presented).

Thus, the Private Base Manager together with the Ginga's modules responsible for retrieving editing commands and data from the several networks supported by Ginga compound the EGA's Update Manager component. All other EGA's components will be defined as NCL <media> element in an NCL EGA document.

NCL editing commands may come from various means. They can be pushed through broadcasting networks, they can be triggered from a user interaction with some receiver's resident application, or they can be produced by an NCL imperative object execution (for example, a Lua script). The first two options can be used to adapt an NCL EGA document. The last option is used by the EGD Structure Constructor to build and update (adapt) an EGD, using its own EGD generated data and also data generated by the EGD Style Constructor.

The independence between the Search Engine and Data manager provided by the Service Data, allow updates of these two components without affecting the NCL EGA and EGD execution. The unique EGA component whose update can affect the EGD presentation is the EGD Structure Constructor. An NCL document presentation is such that any layout change does not affect an already started presentation. Thus, an update of the EGD Style Constructor does not affect the NCL EGA and the NCL EGD execution either. However, to restart the EGD execution with the new layout, the same procedure for the EGD Structure Constructor update must be followed.

To update the EGD Style/Structure Constructor, at first NCL editing commands must be sent stopping the current execution of the EGD. A link relationship among the EGD, the EGD Style Constructor and the EGD Structure Constructor makes the last two to have their execution aborted when the first one is stopped. Then an NCL editing command must be sent to include the new EGD Style/Structure Constructor. Other NCL editing commands must be sent to start the new version of the pair (EGD Style Constructor, EGD Structure Constructor). The EGD Structure Constructor will then restart the EGD creation.

The next section discusses an EPG implementation using the procedures described in this section.

4 An EPG implementation in NCL-LUA

In this proof of concept, a simple architecture instantiation for an EPG application integrated to the Ginga-NCL (Soares et al., 2007) middleware was implemented. In this simple EPG, facilities provided by the Feeding History and Component Persistency (see Figure 1) were not implemented.

The declarative middleware Ginga-NCL is split into two logical subsystems: Ginga-NCL Presentation Engine and Ginga Common Core (Ginga-CC). The Ginga-CC is responsible for providing basic services relating the receiver platform to the Ginga-NCL Presentation Engine and resident applications. Some of these services useful for the EPG implementation will be stressed in the paper. The Ginga-NCL Presentation Engine is a logical subsystem capable of starting and controlling NCL applications.

The NCL EGA implementation runs on the receiver as a resident NCL application that can receive components for updating on demand or by broadcasting. To meet the Brazilian standard specifications, which define Lua (ABNT NBR 15606-2, 2007) as a scripting language for NCL, a Lua Engine is attached to the Ginga-CC. The NCL EGA implementation has an NCLua object (an NCL <media> element with Lua code) that

implements the Data Manager component (see Figure 1). This NCLua object handles metadata gathered by another NCLua object that implements the Search Engine component (see Figure 1). This new NCLua object uses specific Ginga-CC services to perform its tasks. That is, the Search Engine component is composed by Ginga-CC modules, which cannot be adapted, and by functions to gather metadata from these modules and from other sources (functions that can be adapted). However, in this proof of concept only functions to gather metadata from Ginga-CC were implemented.

In the Ginga-CC, a Tuner module is responsible for receiving transport streams transmitted by content providers. Metadata can be pushed by broadcasters (SI tables multiplexed with other contents in an MPEG-2 Transport Stream) or obtained on demand from service providers. In the first case, SI tables are obtained through using the Data Processing module of the Ginga-CC. In the second case, a Transport component is in charge to control network protocols and interfaces. In Figure 2 the NCLua Search Engine code span obtains metadata (or event information) through the Data Processing module of Ginga-CC and passes the metadata to the Data Manager NCLua object.

Figure 2 NCLua code span part of the search engine's set of instructions

```
1:   event.register(
2:    function (evt)
3:     if evt.class=='si' and evt.type=='epg' then
4:       event.post('out', {
5:         class    = 'ncl',
6:         type     = 'attribution',
7:         property = 'sedm',
8:         action   = 'stop',
9:         value    = tableToString(evt)
10:      })
11:    end
12:   end
13:  )
14:
15:  local date1 = os.date'*t'
16:  local date2 = {}
17:  for k,v in pairs(date1) do
18:      date2[k] = v
19:  end
20:  date2.hour = date1.hour + 3
21:  event.post('out', {
22:      class     = 'si',
23:      Type      = 'epg',
24:      Stage     = 'schedule',
25:      startTime = date1,
26:      endTime   = date2
27:  })
```

In Figure 2, line 1 registers the NCLua object to receive events. If the events come from SI tables (evt.type = = 'epg'), the event is passed to the Data Manager component as a result of an NCL attribution action (Soares and Rodrigues, 2006), as specified in lines 2 to 12. Lines 15 to 20 delimit the time interval for SI events to be received. Finally, lines 21 to 27 commands the Ginga-CC to report SI events in the defined time interval.

The NCLua Search Engine and Data Manager objects communicate via an NCL <link> element. This element allows a data to be sent to a <property> element of the NCLua Data Manager object when an NCL attribution action (see line 2 to 12) is issued by the NCLua Search Engine object.

Indeed, any communication between all EGA's components is realised through NCL <link> elements. In all cases, an attribution action issued by the source of the communication causes the parameter passing to a <property> element of the destination NCLua object.

The implemented NCLua Data Manager is very simple and does not perform any data filtering, passing the metadata received from the Search Engine directly to the Service Data component. The only computation realised is fitting metadata to the internal format used in the Service Data.

The Service Data is a table also implemented as an NCLua object. Every time NCLua Data Manager sets a value to a <property> element of NCLua Service Data, this value is put into the table and passed to the NCLua object that implements the EGD Structure Constructor. The value could also be passed directly from the Data Manager to the EGD Structure Constructor, but anyway it must be put into the Service Data table. This is because an update to the NCLua EGD Structure Constructor object will require the access to previously received metadata, which are stored in the Service Data table.

When the NCLua EGD Structure Constructor object has its <property> element value set by the Service Data, it creates new NCL <media> elements in the NCL EGD and can create new navigational relationships (NCL <link> elements) with other EGD <media> elements, to produce the data structure needed to present received metadata and to relate them to other metadata. All these NCL EGD elements are created by using NCL live editing commands (ABNT NBR 15606-2, 2007). If necessary for presentation, Lua functions of the NCLua EGD Style Constructor object is triggered to define the layout and player for the new created EGD <media> elements. If the EGD does not exist when the NCLua EGD Structure Constructor object has its <property> element value set by the Service Data, the NCL EGD is created and started, also using NCL live editing commands.

Figure 3 shows a code span of the NCLua EGD Structure Constructor object. Line 3 tests if a stop action was issued. Line 4 tests if this action came from setting a value (metadata) to the 'sdesc' <property> element, triggered by the Service Data. Line 5 tests if there was an NCL EGD already created. If no, an EGD is created and started through NCL live editing commands issued by the *ega:createEGD* Lua function, as shown in line 6. If yes, received metadata (evt.value) are converted into NCL <media> elements (and, if necessary, NCL <link> elements) to be presented by the NCL Formatter; these elements are created through NCL live editing commands issued by the *ega:processEvent* Lua function, as shown in line 8.

Figure 3 NCLua code span part of the EGD Structure Constructor's set of instructions

```
1:  event.register(
2:   function (evt)
3:    if evt.class=='ncl' and evt.action=='stop' and
                    evt.type=='attribution' then
4:      if evt.property == 'sdesc' then
5:        if ~ega:hasEGD() then
6:          ega:createEGD()
7:        end
8:        ega:processEvent(stringToTable(evt.value))
9:      end
10:   end
11:  end
12: )
```

The layout and player for EGD <media> elements are created by the NCLua EGD Style Constructor object through defining new NCL <descriptor> and <region> elements, also by using the NCL live editing commands. NCL <descriptor> elements can also define navigational relationships among metadata.

All NCLua objects are children of the NCL document that implements the EGA, as shown in Figure 4.

Figure 4 NCL EGA Implementation (see online version for colours)

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

<head>
  <ruleBase>
   <rule id="seRule" var="service.defaultSearchEngine"
    comparator="eq" value="false"/>
   ...
  </ruleBase>
   ...
</head>

 <body>
  <port id="sePort" component="searchEngineSwitch"/>
   ...
  <switch id="searchEngineSwitch">
   <bindRule constituent="newSearchEngine" rule="seRule"/>
   <defaultComponent component="defaultSearchEngine"/>
   <media id="defaultSearchEngine" src="searchEngine.lua"/>
   <media id="newSearchEngine" src="newSearchEngine.lua"/>
  </switch>
   ...
</body>
</ncl>
```

In Figure 4, the NCL specification initially defines a <ruleBase> element that specifies a set of rules to be used in tests to select the default implementation (the NCLua objects previously described) or a new created substitute component. Four rules are defined to allow adaptations of four components: a <rule> element (*id*='seRule' in the figure) to allow Search Engine adaptations, and other three analogous <rule> elements to allow Data Manager, EGD Style Constructor, and EGD Structure Constructor adaptations.

The NCL EGA execution entry points are defined by four <port> elements, children of the <body> element. These four elements are mapped to <switch> elements. In Figure 4, the <port *id*='sePort'> element and the <switch *id*="searchEngineSwitch"> element are shown. The <switch> elements group <media> elements to be chosen, in agreement with previously defined <rule> elements, referred in the <switch> by its child <bindRule> element. Also in Figure 4, the <switch *id*="searchEngineSwitch"> element specifies that if the <rule *id*='seRule'> element is satisfied, the Search Engine component must be updated, otherwise the default component (the previously NCLua object that contains the code span of Figure 2) must be chosen.

Therefore, to adapt the EGA's Search Engine component, an NCL live editing command must be sent stopping the current execution of the <switch id="searchEngineSwitch"> element. Then an NCL live editing command must be issued to include the new <media id="newSearchEngine"> into the same stopped <switch> (if the new object already exists it will be substituted). Afterwards, another NCL live editing command must be issued to set the service.defaultSearchEngine variable to 'true'. Finally, another NCL live editing command must be issued to restart the <switch id="searchEngineSwitch"> element execution.

5 Final remarks

This paper presents architecture for building componentised EPG/ESG applications. In the proposed architecture software components can be adapted in real time without affecting the EPG/ESG presentation. Not only the architecture's components can be adapted on-the-fly but also the meta-information presented to users. The several software components are responsible for performing adaptations during the processing phases of the meta-information.

The EPG/ESG software components can be implemented part as a resident application, taking advantage of the performance offered by a particular platform, and other part using components received by broadcast or on demand.

As a proof of concept a simple adaptive EPG was implemented in Ginga-NCL, the middleware of the Brazilian Terrestrial Digital TV System. The implementation has shown how easy is to implement an adaptive EPG/ESG in a declarative middleware that uses interpreted scripting languages.

Not only the proposed modular architecture and support to adaptations but also the declarative approach to build EPG/ESG can be considered the main contributions of the paper.

As future work, the described implementation is being extended to also include the History Manager and Component Persistency components. The first one will provide support to a recommender system also in development. The second one will make the <switch> element a real set of component alternatives. Besides the default implementation, other alternatives may be resident or may be made persistent for future selection.

The Search Engine component implementation is being also extended to a new version that will incorporate access to other metadata providers besides metadata sent by broadcast in SI tables. This new version will also accept query to specific metadata information retrieval.

A new Data Manager version is in development. On the basis of the content analysis of the TV programme or service being received, this new version will be able to command the Search Engine to find new metadata and services related to the current exhibition. This new version will also be able to make metadata content adaptations based on user preferences.

New EGD Style Constructor and EGD Structure Constructor components are also in development to generate structured EPG/ESG to be presented both by NCL and SMIL players using multiple exhibition devices. The idea is to have different EPG/ESG running in different devices while sharing the same programme/service exhibition screen.

References

ABNT NBR 15603-1 (2007) *Digital Terrestrial Television – Multiplexing and Service Information (SI) Part 1: SI for Digital Broadcasting Systems*, ABNT Standard, November, ISBN 978-85-07-00610-7, ABNT NBR 15603-1:2007 41 pages.

ABNT NBR 15606-2 (2007) *Terrestrial Digital TV – Data Coding and Transmission Specification for Digital Broadcasting – Part 2: Ginga-NCL for Fixed and Mobile Receivers: XML Application Language for Application Coding*, ABNT Standard, September, ISBN 978-85-07-00605-3, ABNT NBR 15606-2:2007, 284 pages.

Ardissono, L., Gena, C., Torasso, P., Bellifemine, F., Chiarotto, A., Difino A. and Negro, B. (2003) *Personalized Recommendation of TV Programs*, Lecture notes in computer science, Vol. 2829/2003, pp.474–486, ISSN 0302-9743 (Print) 1611-3349 (Online).

ARIB Standard STD-B24 (2005), *Data Coding and Transmission Specifications for Digital Broadcasting*, ARIB Standard STD-B24, Vol. 2, February.

ARIB STD-B10 (2006) *Service Information for Digital Broadcasting System*, ARIB Standard, 28 September.

ATSC A/65b (2003) *Program and System Information Protocol*, ATSC Standard, 18 March.

DVB Document A038 Rev. 3 (2007) *Specification for Service Information (SI) in DVB Systems*, DVB Standard, July.

Hjelm, J. (2008) *Why IPTV: Interactivity, Technologies, and Services*, John Wiley and Sons, Ltd., http://www.amazon.com/Why-IPTV-Interactivity-Technologies-Explained/dp/0470998059

Ierusalimschy, R. (2008) *Programming in Lua*, Lua.org.

Morris, S. and Smith-Chaigneau, A. (2005) *Interactive TV Standards*, Elsevier Inc., http://www.amazon.com/gp/product/0240806662/ref=s9_simz_gw_s0_p14_i1?pf_rd_m=ATVPDKIKX0DER&pf_rd_s=center-2&pf_rd_r=085X534Y9CDQWV5T0FWW&pf_rd_t=101&pf_rd_p=470938631&pf_rd_i=507846

O'Sullivan, D., Smyth, B., Wilson, D.C., Mcdonald, K. and Smeaton, A. (2004) '*Improving the Quality of the Personalized Electronic Program Guide, User Modeling and User-Adapted Interaction*', User Modeling and User-Adapted Interaction, Vol. 14, No. 1, pp.5–36, February, ISSN: 0924-1868.

SMIL (2008) *Synchronized Multimedia Integration Language (SMIL 3.0)*, W3C Recommendation 1 December, Avaiable at: http://www.w3.org/TR/SMIL3/

Smyth, B. and Cotter, P. (2001) 'Personalized electronic programme guides', *Artificial Intelligence Magazine*, Vol. 22, No. 2, pp.89–98.

Soares, L.F.G. and Rodrigues, R.F. (2006) *Nested Context Language 3.0: Part 8 – NCL (Nested Context Language) Digital TV Profiles*, Monografias em Ciência da Computação do Departamento de Informática, PUC-Rio, No. 35/06, Rio de Janeiro, October, ISSN 0103-9741.

Soares, L.F.G., Rodrigues, R.F. and Moreno, M.F. (2007) 'Ginga-NCL: the declarative environment of the Brazilian digital TV system', *Journal of the Brazilian Computer Society*, Vol. 12, No. 4, pp.37–46, March. ISSN 0104-6500.

Vaguetti, L. and Gondim, P. (2008) 'A web-service-based architecture solution to ubiquitous personalized digital television content', *5th International Workshop on Ubiquitous User Modeling*, Gran Canaria, Spain.

Notes

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

[2]MythTV project: http://www.mythtv.org

[3]The SBTVD (the Brazilian DTV standard) standard does not require SMIL support. However, the reference implementation of Ginga for SBTVD allows SMIL Tiny profile documents embedded in NCL documents.

[4]This facility could not be used in SMIL EGD, since SMIL does not have defined any support for live editing. In case of using SMIL the EGD must be rebuilt periodically.