MAESTRO: The Declarative Middleware Proposal for the SBTVD

TeleMidia Laboratory
PUC-Rio
R. Marquês de São Vicente, 225
Rio de Janeiro – 22453-900 – Brazil
lfgs@inf.puc-rio.br

MAESTRO: The Declarative Middleware Proposal for the SBTVD

TeleMidia Laboratory
PUC-Rio
R. Marquês de São Vicente, 225
Rio de Janeiro – 22453-900 – Brazil
lfgs@inf.puc-rio.br

Abstract

This paper deals with the declarative middleware proposal for the SBTVD. The middleware, named MAESTRO, is based on the Nested Context Language (NCL) and was implemented in C++ and also in Java. It can run alone in a set-top box, or integrated with a procedural middleware, or even run as an XLet.

1. Introduction

The Brazilian Digital TV System (SBTVD) is under definition and it is expected to be launched in the end of 2006. Conceived for terrestrial ("free-to-air") TV, the system should take into account all other system reference models without neglecting the topographic, political and social peculiarities of the country and its people. Moreover, SBTVD should take profit of being conceived when emerging technologies previously unfeasible is now available.

This paper deals with the declarative middleware proposal for the SBTVD. The middleware, named MAESTRO, is based on the Nested Context Language (NCL) [1] and have its main parts available as open source software (C++ version and also a Java version), under GNU General Public License [2].

2. An Overview

In 1997, the ISO Multimedia Experts Group (MHEG) published the MHEG standard. The concept of nested compositions was incorporated by MHEG since its 1992 meeting in Brazil, from a NCM (the model behind NCL) proposal. MHEG-1 included support for objects that contained procedural code, extending the basic MHEG-1 model. The MHEG-3 standard defined a standardized virtual machine and byte code representation that allowed this code to be portable across hardware platforms.

"MHEG-1 and MHEG-3 were not very successful, partly because the underlying concepts were very complicated to date and because the industry was not yet ready for the features offered by these standards. MHEG-3 was overtaken by the success of Java, and thus in 1998 MHEG-6 added support for using Java to develop script objects, thus mixing the declarative strengths of MHEG with the procedural elements of Java"[3]. However, Java objects had to be called from their MHEG parent applications.

MHP was the first open middleware standard based purely on Java, meaning that receivers did not need to implement another technology to use it. This shift from declarative approaches toward Java does not mean that the declarative approaches failed. Declarative languages are usual higher level and conceived for a particular application domain. When they match with the application focus they are perfectly adequate, and may actually be superior to procedural languages. Declarative technologies certainly have their place, and these are included in the ISDB, ATSC and DVB middlewares using HTML and plug-ins for non-Java application formats.

Non-linear TV programs are the most important applications in the interactive TV environment. They are usually composed by the main audio and video TV-program streams together with other media objects synchronized in time and space. In this scenario, user interaction is just a special case of temporal synchronization. Thus, a declarative language for supporting non-linear programs must focus on synchronization in general and not on user interaction in particular. Declarative languages like MHEG, NCL, SMIL, XMT-O follow this approach.

A declarative middleware should include support for objects that contain procedural code, which could extend its model to add decision-making features that were otherwise not possible. This is crucial in low cost set-top boxes where the embedded procedural language must be very simple in order to not burden the device.

A declarative middleware should also be able to be incorporated (peer-to-peer) in a procedural middleware, taking profits of all procedural facilities, as well as being able to be launched by procedural applications that needs the facilities for which the declarative language was conceived.

MAESTRO fills all these requirements as presented in the next sections.

3. NCL 2.3

Conceptually, is NCL set out to do for non-linear TV programs what HTML did for WWW documents. The central point is that HTML has its focus on user interactions while NCL has its focus on media object synchronization where user interactions are treated as a special case of non-deterministic synchronizations.

As usual in XML based language, NCL 2.3 follows a modular approach. Its Structure module defines the basic structure of an NCL document. It defines the root element, called *ncl*, the *head* and the *body*, following the terminology adopted by other W3C standards.

The Layout module specifies elements and attributes that define how objects will be presented inside regions of output devices.

The Media module defines basic media object types. Presentation and selection events (occurrences in time) can be defined over media object contents. Attribution events can be defined over media object attributes. Events are used for spatial and temporal synchronization defined by *links*. Besides the usual *animation*, *audio*, *image*, *text*, *textstream*, and *video* media objects, NCL 2.3 defines the *setting* (grouping attributes that defines the environment), *time* (that represents the time), *NCLua* (procedural object with LUA code), and *NCLet* (procedural object with Java XLet code) objects that act like the other usual objects with regard to temporal and spatial relationship definitions. Furthermore, an HTML *text* object can be defined, meaning that NCL 2.3 treats a HTML document (in its plenitude) as just one of its objects. Therefore, MAESTRO can run any HTML application defined by ISDB, DVB and ATSC, depending only on the HTML player implementation.

The Context module is responsible for defining *context* nodes. An NCL context node contains a set of nodes and a set of links. Contexts do not have temporal semantics like *seq* and *par* elements of SMIL and XMT-O. Instead, they can get these semantics and other ones more sophisticated, when used together with the Template module.

MediaContentAnchor module allows media object *content-anchor* definition, used to define presentation and selection events. CompositeNodeInterface module allows composite node *content-anchor* definitions for composite nodes, and also *port* definitions, which exports child node interfaces (*ports*, *attributes* or *anchors*), allowing relationships (defined by *links*) to go inside a composite node without losing compositionality. AttributeAnchor module allows the definition of node *attributes* as node interfaces, used to define attribution events. SwitchInterface module allows the definition of special interfaces for *switch* elements, which will be mapped to a set of alternative interfaces of internal switch nodes, allowing a link to anchor on the component chosen when the *switch* is processed.

The XConnector module allows defining *connectors*, which can be used to specify reference and spatio-temporal synchronization relations (*causal* and *constraint* relations), treating reference (user interaction) relations as a particular case of synchronization relations. The Linking module is responsible for defining *links* (relationships) using *connectors* (relations). A *link* must refer to a *connector* and to events that will participate in the relationship.

The Descriptor module allows decoupling a media object from its presentation specification, allowing an object to have multiple different presentations. A *descriptor* element has temporal attributes defined by the Timing module, which allows specifying what will happen with an object at the end of its presentation, its ideal duration, and its allowed duration-variation (and associated cost). Another descriptor attribute identifies the presentation tool to be used (which can be parameterized by descriptor child elements). Still another attribute refers to a region defined by elements of the Layout module. So, a region is associated with an object by a descriptor. When a language profile uses the Descriptor module, it has to determine how *descriptors* will be associated with document components. NCL 2.3 Language Profile allows a *descriptor* to be associated with any media or context node, through *media* and *context* elements attributes, or through *link* endpoint or through parent *composite* node attributes.

The TestRule module allows the definition of test rules that, when satisfied, select alternatives for document presentation. The ContentControl module specifies the *switch* element, allowing the definition of alternative document nodes to be chosen during presentation time. The DescriptorControl module specifies the *descriptorSwitch* element, which contains a set of alternative descriptors, chosen during presentation time, to be associated with an object. It should be remarked that *descriptorSwitches* may be associated with *media* and *context* elements similar to *descriptor*s.

NCL 2.3 allows intensive reuse of its elements. The NCL 2.3 Reuse functionality is partitioned into two modules: Import and ReuseEntity. Import module allows importing connector bases, descriptor bases, region bases, cost function bases, test rule bases, and template bases from other NCL documents. Moreover,

the module allows importing an entire NCL document to another NCL document. The ReuseEntity module allows an NCL element to be reused. An element to be reused must be referred by another element, and they must be considered the same. When a language profile uses this module, it may refer to an element inside the NCL document itself or inside an imported base or document.

The Metainformation module allows the definition of metadata about documents and about their elements, following the RDF – Resource Description Framework recommendation.

Finally, the XTemplate module allows the definition of composite-node templates. A composite-node template has a *vocabulary* and, optionally, *constraints* on the vocabulary. The *vocabulary* defines types of components, types of connectors (relations) and interfaces for a composite node whose structure is inherited from the template; *constraints* define restrictions on vocabulary elements and on the inclusion of instances of components and connectors, without identifying who all components and relationships are. This task is the responsibility of the specific composition using the template. Templates may be used to give temporal or spatial semantics to a context element and are very important in easing the authoring of NCL documents. Templates of arbitrary different semantics may be created and then easily used in the document authoring. For example, a template base may include the definition of templates with the same semantics of the *par* and *seq* elements of SMIL. Importing this base, an NCL document can be authored as if it had *par* and *seq* elements and act just like SMIL.

4. MAESTRO Architecture

Figure 1 shows the main modules of the MAESTRO architecture. The current implementation has one version in C++ and another one (identical) in Java. The java version is integrated with a procedural Java middleware of the SBTVD. Both version runs using a Linux like OS.

The MAESTRO core is the NCL formatter, which is in charge of receiving the NCL document and controlling its presentation, trying to guarantee that specified relationships among media objects are respected. The formatter must instantiate media players when necessary, passing to them media contents received via the MPEG system transport stream (via DSM-CC or regular MPEG elementary streams), or via the return channel (TCP/IP). The formatter must receive event notification through the players and through the MPEG system transport stream (stream events), in the case of events generated in live transmissions.

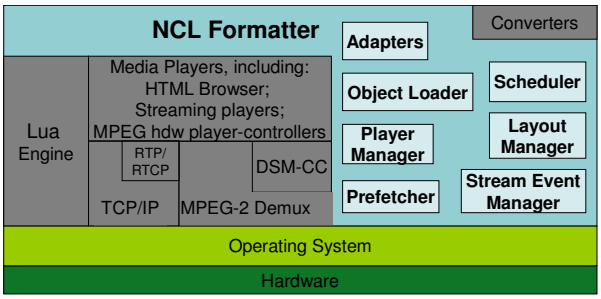

Figure 1 - MAESTRO Architecture

Media players (presentation tools) are modules that can be plugged in the *NCL formatter* by using an opened interface specified to not only allow the incorporation of external (legacy) presentation tools, but also to enable them to interact with each other. Players that do not conform to the formatter API must join the system through adapters. Players are implemented in software and in hardware (for MPEG video and audio main stream, for example).

It is important to note that the HTML browser is one of the several NCL players. This means that any declarative application developed for HTML will run over MAESTRO.

Another special player is the Lua Engine. Lua [4] is a powerful light-weight programming language designed for extending applications. Lua combines simple procedural syntax with powerful data description constructs based on associative arrays and extensible semantics. Lua is dynamically typed, interpreted from bytecodes, and has automatic memory management with garbage collection, making it ideal for configuration, scripting, and rapid prototyping.

Although not shown in Figure 1, MAESTRO can also play NCLet objects when supported by a Java virtual machine. NCLet object has an XLet as its content.

Finally, MAESTRO is able to receive non-linear programs specified in other languages than HTML and NCL. Converters are developed to translate XMT-O and SMIL specifications into NCL specifications and vice-versa.

5. MAESTRO Interoperability

There are several alternatives for the procedural and declarative middleware of the SBTVD. As regarding MAESTRO, three alternatives must be stressed.

The first one, shown in Figure 2, is a resident middleware with only the declarative module. Procedural codes could be launched from declarative applications. As aforementioned, MAESTRO allows procedural objects in NCL documents: XLets, ECMA Scripts and Lua codes. Applications where synchronization is the focus are advantaged by this alternative.

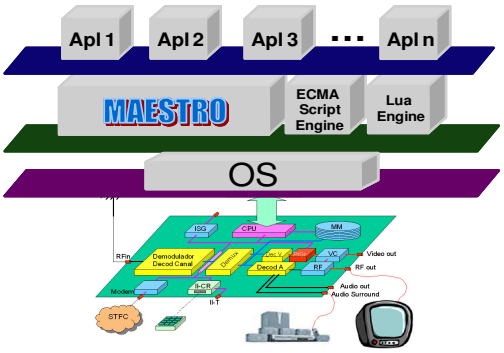

Figure 2 – SBTVD Middleware: MAESTRO alone

The second alternative, shown in Figure 3, is Maestro coupled with a procedural middleware (using Java in the SBTVD case). Of course this alternative will require a set-top box with better performance. However, a better support for all kind of applications is achieved.

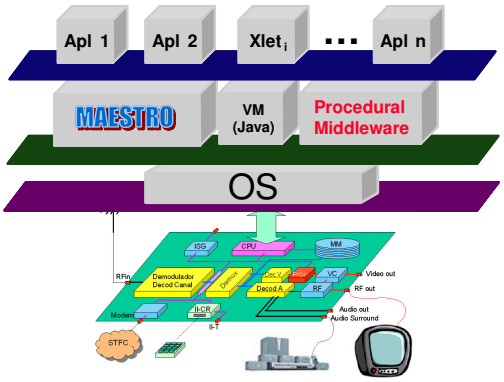

Figure 3 - MAESTRO + Procedural Middleware

It must be mentioned that both alternatives already presented can run not only NCL but also HTML applications. This means that any content developed for the three main digital TV systems (ISDB, ASTC and DVB) is supported by MAESTRO.

The third alternative runs MAESTRO triggered by a procedural middleware in Java, as shown in Figure 4. MAESTRO was also implemented as an XLET that can be exported to the user set-top box. Therefore, any application developed in NCL will be able to run in any of the main digital TV systems.

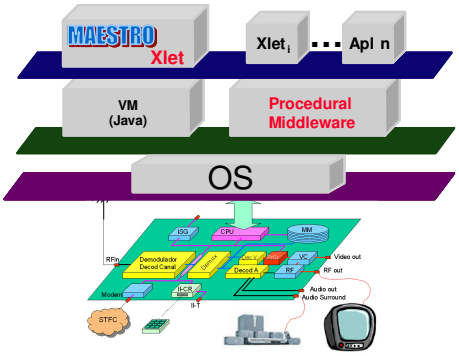

Figure 4 - MAESTRO as an XLet

6. Conclusions

MAESTRO prototype (C++ and Java version) is operational since June 2005. Several non-linear programs were developed using NCL, stressing all its facilities. A graphical NCL editor was also developed as open source software under GNU General Public License [5].

7. REFERENCES

[1] Soares L.F.G., Rodrigues R.F. "Nested Context Model 3.0. Part 5 – NCL (Nested Context Language)". *Technical Report MCC - 26/05, PUC-Rio*. Rio de Janeiro. June 2005. ISSN 0103-9741.

[2] NCL Formater. TeleMidia Laboratory. http://opensource.telemidia.puc-rio.br/maestro/nclformatter/ or http://opensource.telemidia.puc-rio.br/hyperprop/nclformatter/ or http://opensource.telemidia.puc-rio.br/projects/nclformatter/

[3] Morris, S.; Chaigneau, A.S. "Interactive TV Standards" Chapter: A History Lesson: The Background of MHP and OCAP". *Focal Press, Elsevier*, 2005. ISBN: 0-240-80666-2.

[4] Ierusalimschy, R.; Figueiredo, L.H.; Celes, W. "Lua 5.0 Reference Manual". *Technical Report MCC -14/03, PUC-Rio*. Rio de Janeiro. May 2003. ISSN 0103-9741.

[5] NCL Editor. TeleMidia Laboratory. http://opensource.telemidia.puc-rio.br/maestro/ncleditor/ or http://opensource.telemidia.puc-rio.br/hyperprop/ncleditor/ or http://opensource.telemidia.puc-rio.br/projects/ncleditor/